Matlab 应用教程丛书

Matlab 图像处理与应用

（第 2 版）

高　成　主编

董长虹　郭　磊　李阳明　副主编

赖志国　陈继云　李淑红　编著

国防工业出版社

·北京·

图书在版编目(CIP)数据

Matlab 图像处理与应用 / 赖志国等编著. —2 版. —北京:
国防工业出版社,2007.4

(Matlab 应用教程丛书/高成主编)

ISBN 978-7-118-04988-6

Ⅰ.M… Ⅱ.赖… Ⅲ.计算机辅助计算-软件包,MATLAB—
教材 Ⅳ.TP391.75

中国版本图书馆 CIP 数据核字(2007)第 021371 号

※

国防工业出版社出版发行

(北京市海淀区紫竹院南路 23 号 邮政编码 100044)

北京奥鑫印刷厂印刷

新华书店经售

*

开本 710×960 1/16 **印张** 18½ **字数** 348 千字

2007 年 4 月第 2 版第 1 次印刷 **印数** 1—4000 册 **定价** 28.00 元

(本书如有印装错误,我社负责调换)

国防书店:(010)68428422 发行邮购:(010)68414474

发行传真:(010)68411535 发行业务:(010)68472764

丛书编委会

前　言

Matlab 图像处理工具箱提供了强大的图像处理、图形界面设计的功能。对于广大的使用者来说，可能存在对各个函数的理论背景模糊不清，不能准确应用各个函数的问题；也可能存在对 Matlab 图像处理工具箱中的函数十分熟悉，但是不能灵活运用来解决实际情况的问题。

本书的内容以实践为基础，简洁明了地指出了所介绍的函数、方法的理论背景，同时又紧密联系实际应用，以具体的实例说明了函数、方法。在实例中强调了如何用 Matlab 解决图像处理中的问题、难题，节省图像处理工作者的时间和精力，提高图像处理的效率。

本书着重于图像处理工具箱的具体应用，通过具体的分析和详细的实例，读者不仅可以对 Matlab 图像处理工具箱函数的强大功能有一个深刻的了解，更能学会正确运用它快速解决实际问题的方法，从而提高分析和解决问题的实际能力。

本书的第 1 章～第 4 章是 Matlab 的使用和图形处理技术，在介绍 Matlab 的基础上，讲解了图形绘制和图形几何操作等技术在 Matlab 中的实现；第 5 章～第 8 章是图像变换以及基于基础图像理论的图像滤波、图像分割、

图像压缩等技术在实际中的应用；第 9 章是前 8 章讲解知识的综合应用，有助于拓宽思路、灵活地应用基本 Matlab 图像处理函数；第 10 章是图形用户界面设计。

本书中的所有程序都经过实际调试。但是由于本书覆盖的领域广泛，内容多，难免有错误和疏漏的地方，欢迎广大读者批评指正。

编 者

2006 年 12 月

目　录

第1章 Matlab概述

1.1 Matlab简介

Matlab语言是当今国际上科学界（尤其是自动控制领域）最具影响力，也是最有活力的软件。它起源于矩阵运算，并已经发展成一种高度集成的计算机语言。它提供了强大的科学运算、灵活的程序设计流程、高质量的图形可视化界面设计、便捷的与其他程序和语言接口的功能。Matlab语言在各国高校与研究单位发挥着重大的作用。

Matlab语言的首创者Cleve Moler教授在数值分析特别是数值线性代数的领域中很有影响，他参与编写了数值分析领域一些著名的著作和两个重要的Fortran程序——EISPACK和LINPACK。他曾在密西根大学、斯坦福大学和新墨西哥大学任数学与计算机科学教授。1980年前后，当时的新墨西哥大学计算机系主任Moler教授在讲授线性代数课程时，发现用其他高级语言编程极为不便，便构思并开发了Matlab（Matrix Laboratory，矩阵实验室），这一软件利用了当时数值线性代数领域最高水平的EISPACK和LINPACK两大软件包中可靠的子程序，用Fortran语言编写了集命令翻译、科学计算于一身的一套交互式软件系统。

交互式语言，是指人们给出一条命令，立即就可以得出该命令的结果。该语言无需像C语言和Fortran语言那样，首先要求使用者去编写源程序，然后对之进行编译、连接，最终形成可执行文件。这无疑会给使用者带来了极大的方便。早期的Matlab是用Fortran语言编写的，只能作矩阵运算；绘图也只能用极其原始的方法，即用星号描点的形式画图；内部函数也只提供了几十个。但即使其当时的功能十分简单，当它作为免费软件出现以来，还是吸引了大批的使用者。

Cleve Moler和John Little等人成立了一个名叫The MathWorks的公司，Cleve Moler一直任该公司的首席科学家。该公司于1984年推出了第一个Matlab的商业版本。当时的Matlab版本已经用C语言作了完全的改写，其后又增添了丰富多彩的图形图像处理、多媒体功能、符号运算以及与其他流行软件的接口功能，使得Matlab的功能越来越强大。

The MathWorks公司于1992年推出了具有划时代意义的Matlab 4.0版本，

并于 1993 年推出了其微机版，可以配合 Microsoft Windows 一起使用，使之应用范围越来越广。1994 年推出的 4.2 版本扩充了 4.0 版本的功能，尤其在图形界面设计方面更提供了新的方法。

1997 年推出的 Matlab 5.0 版本允许了更多的数据结构，如单元数据、数据结构体、多维矩阵、对象与类等，使其成为一种更方便编程的语言。1999 年初推出的 Matlab 5.3 版本在很多方面又进一步改进了 Matlab 语言的功能。

2000 年 10 月底推出的 Matlab 6.0 正式版(Release12)，在核心数值算法、界面设计、外部接口、应用桌面等诸多方面有了极大的改进。

2004 年 9 月，The MathWorks 公司新推出了 Matlab 7.0 (Release 14) 版本，主要包括 12 个新产品模块，同时主要升级了 28 个产品模块。Matlab 7.0 针对编程环境、代码效率、数据可视化、数学计算、文件 I/O 等方面进行了升级，具体如下。

开发环境 ：

- 重新设计的桌面环境，针对多文档界面应用提供了简便的管理和访问方法，允许用户自定义桌面外观，创建常用命令的快捷方式；
- 增强数组编辑器 (Array Editor) 和工作空间浏览器 (Workspace Browser) 功能，用于数据的显示、编辑和处理；
- 在当前目录浏览器 (Current Directory Browser) 工具中，增加代码效率分析、覆盖度分析等功能；
- M－Lint 编码分析，辅助用户完成程序性能分析，提高程序执行效率；
- 增强 M 文件编辑器 (M－Editor)，支持多种格式源代码文件可视化编辑，如 C/C＋＋、HTML、Java 等。

编程：

- 支持创建嵌套函数 (Nested Function)，提供更灵活的代码模块化方式；
- 匿名函数 (Anonymous Function) 功能，支持在命令行或者脚本文件中创建单行函数 (Single Line Function)；
- 支持条件分支断点，可以在条件分支语句中进行程序中断调试；
- 模块化注释，支持为代码段注释。

数学：

- 支持整数算术运算；
- 支持单精度数据类型运算，包括基本算术运算、线性代数、FFT 等；
- 使用更强大的计算算法包 Qhull 2002.1 ，提供更丰富的算法支持；
- linsove 函数用于处理线性代数方程求解；
- ODE 求解器能够处理隐性微分方程组以及多点边界问题。

图形和 3D 可视化：

- 新图形窗体界面；
- 直接从图形窗体生成 M 代码，可以完成用户自定义绘图；
- 增强图形窗体注释；
- 数据侦测工具（Data Exploration Tools）提供丰富的数据观测手段；
- 自定义图形对象，提供丰富的图形显示能力；
- GUIDE 新增对用户界面面板和 ActiveX 控件支持；
- 增强句柄图形对象支持完整的 TeX 和 LaTeX 字符集。

文件 I/O 和外部接口：

- 新增文件 I/O 函数，支持读取任意格式文本数据文件，并且支持写入 Excel 和 HDF5 格式数据文件；
- 具有压缩功能的 MAT 文件格式，支持快速数据文件 I/O 能力；
- javaaddpath 函数可以无需重新启动 Matlab 便完成 Java 类的加载、删除等功能；
- 支持 COM 、服务器事件以及 VBS ；
- 支持 SOAP，使用网络服务；
- FTP 对象可直接访问 FTP 服务器；
- 支持 Unicode 编码格式，增强 MAT 文件字符集；
- 性能与系统平台支持；
- JIT 加速器支持所有数值数据类型；
- Windows XP 系统下支持 3GB 内存访问。

大系统建模：

- 支持将大系统模型分割为不同的文件，每个文件为独立的系统模型；
- 支持系统不同模型文件独立仿真测试；
- 增强系统集成手段，支持配置管理和版本控制软件；
- 递增式模型加载与代码生成功能；
- 针对大模型系统提高运行速度和效率；
- 模型工作空间（Model Workspace），即每个模型都用于独立的工作空间，进行参数管理和数据处理；
- 增强总线支持。

Simulink 和 Stateflow：

- 统一的模型浏览器（Model Explorer），用于浏览、维护、配置、搜索、定义所有模型中相关的参数、数据对象和属性；
- 统一的仿真和代码生成选项；
- 支持创建并保存多种仿真和代码生成选项；
- 数据管理和可视化；

- 新增数据对象属性；
- 可选数据记录增加测试点，无需在模型中增加额外的模块；
- I/O 管理，可以将必要的信号源和信宿连接到模型而不需要增加模块。

Matlab 支持：

- 使用嵌入式 Matlab（Embedded Matlab）功能引入算法并支持 C 代码生成；
- 增强 M 语言 S 函数的支持。

虽然 Matlab 语言是计算数学专家倡导并开发的，但其普及和发展离不开自动控制领域学者的贡献。甚至可以说，Matlab 语言是自动控制领域学者和工程技术人员捧红的，因为在 Matlab 语言的发展进程中，许多有代表性的成就和控制界的要求与贡献是分不开的。迄今为止，大多数工具箱也都是控制方面的。Matlab 具有强大的数学运算能力、方便实用的绘图功能及语言的高度集成性，它在其他科学与工程领域的应用也是越来越广，并且有着更广阔的应用前景和无穷无尽的潜能。如果有一种十分有效的工具能解决在教学与研究中遇到的问题，那么 Matlab 语言正是这样的一种工具。它可以将使用者从繁琐、无谓的底层编程中解放出来，把有限的宝贵时间更多地花在解决问题中，这样无疑会提高工作效率。

目前，Matlab 已经成为国际上最流行的科学与工程计算的软件工具，现在的 Matlab 已经不仅仅是一个“矩阵实验室”了，它已经成为一种具有广泛应用前景的全新的计算机高级编程语言了，有人称它为“第四代”计算机语言，它在国内外高校和研究部门正扮演着重要的角色。Matlab 语言的功能也越来越强大，不断适应新的要求并提出新的解决方法。可以预见，在科学运算、自动控制与科学绘图领域，Matlab 语言将长期保持其独一无二的地位。

Matlab 工作环境包括：帮助系统、工作内存管理、指令和函数管理、搜索路径管理、操作系统、程序调试和性能剖析工具等。Matlab 改变了过去单调依靠“在指令窗通过纯文本形指令进行各种操作”的面貌，引入了许多让使用者一目了然的图形界面，如在线帮助的交互式界面 helpwin、管理工作内存的 workspace、交互式的路径管理界面 pathtool、指令窗显示风格设置界面等，它们的开启方式有：工具条图标开启、选择菜单项开启和直接“文本式”指令开启。在 Matlab 5.0 以后的版本中更进一步把图形显示窗改造成了交互操作的可编辑图形界面。

进入 Matlab 之后，会看到一个窗口：MATALB，称为指令窗口（见图 1.1），它是键入指令的地方，也是 Matlab 显示计算结果的地方。该指令窗口的功能选单一共有 File、Edit、Debug、Desktop、Window，Help 六个主要功能，每一个主要功能之下又各有下一层的功能，这些将在后面的内容中结合使用陆续说明。

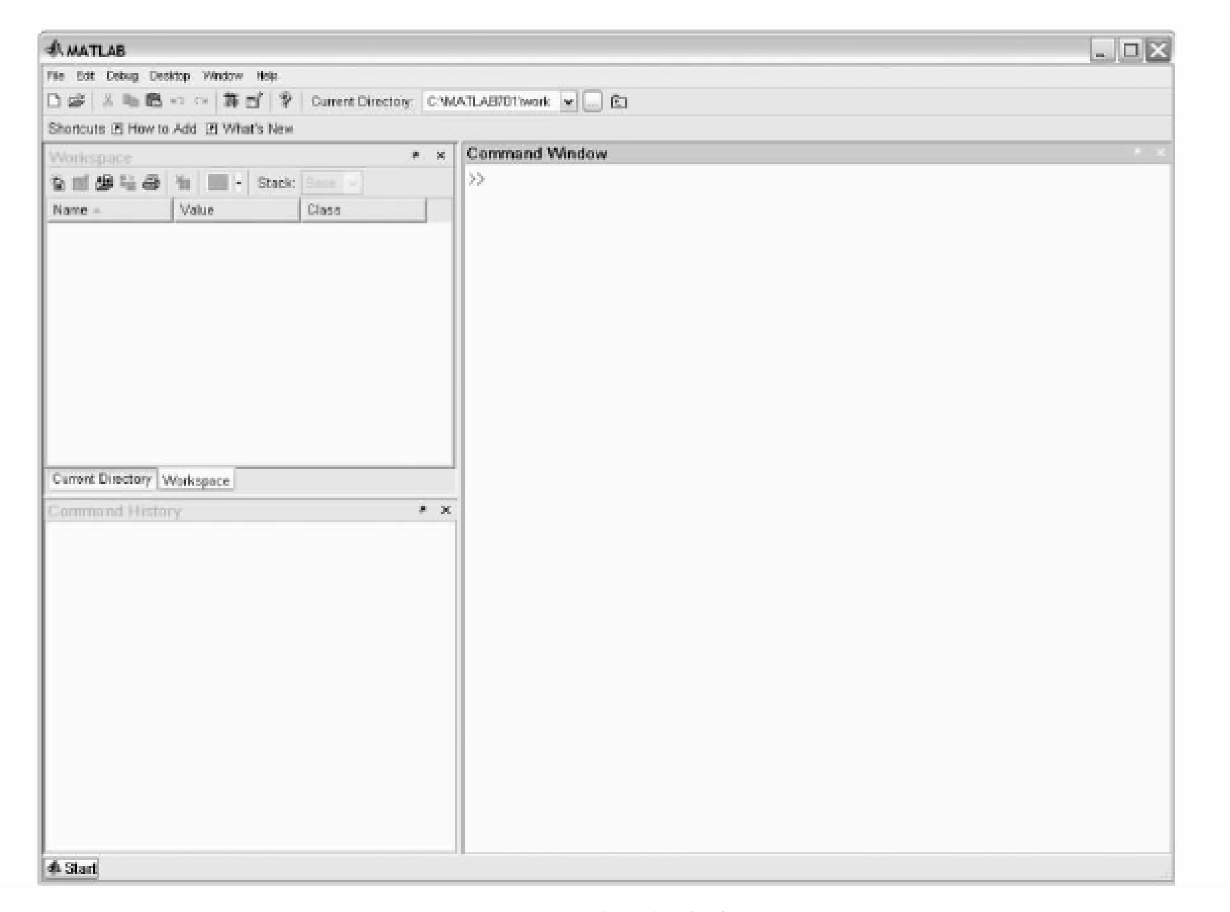


图 1.1 Matlab 指令窗口

1.2 Matlab 编程基础

1.2.1 变量和数学运算

首先从 Matlab 的数学运算开始说明。例如：要算 $1+2+3$ 及 $1\times10+2\times20+3\times30$ 这两个式子，以下的例子中接着提示符号“＞＞”之后的是要键入的算式，Matlab 将计算的结果以 ans 显示。如果算式是 $x=1+2+3$，Matlab 将计算的结果以 x 显示。

```
>> 1+2+3
ans=
6
>> 1*10 + 2*20 + 3*30
ans=
140
>> x=1+2+3
x =
6
```

如果在上述的例子结尾加上“;”，则计算结果不会显示在指令窗口中，要得知

计算值只需键入该变量即可：

```
>> x=1+2+3;
>>x
x=
6
```

下面的例子显示了 Matlab 使用变量的灵活性。

```
>> apple=5
apple=
5
>> orange=10
orange=
10
>> total_cost=apple*2+orange*4
total_cost=
50
>> average_cost=total_cost/(apple+orange)
average_cost=
3.33334
```

Matlab 提供基本的算术运算有：加(+)、减(-)、乘(*)、除(/)、幂(^)。如下面的语句：

```
5+3, 5-3, 5*3, 5/3, 5^3
```

其他在计算中常用的功能以一个算式来说明。例如，要计算面积 area$=\pi r^2$，半径 $r=2$，则可键入：

```
>>r=2;
>>area=pi*r^2;
>>area=
12.5664
```

也可以将上述指令打在同一行，以“,”或“;”分开，例如：

```
>>r=2,area=pi*r^2;
>>r=2;area=pi*r^2;
```

注意上述二式的差异，前者有计算值显示，而后者则无。如果一个指令过长，可以在结尾加上“...”(代表此行指令与下一行连续)，例如：

```
>>r=2;
>>area = pi...
*r^2
```

Matlab 提供了众多的基本数学函数，其分类列表见表 1.1～表 1.6，这些函数在以后的章节中都会用到。

表 1.1 三角函数和双曲函数

名称	含 义	名称	含 义	名称	含 义
sin	正弦	csc	余割	atanh	反双曲正切
cos	余弦	asec	反正割	acoth	反双曲余切
tan	正切	acsc	反余割	sech	双曲正割
cot	余切	sinh	双曲正弦	csch	双曲余割
asin	反正弦	cosh	双曲余弦	asech	反双曲正割
acos	反余弦	tanh	双曲正切	acsch	反双曲余割
atan	反正切	coth	双曲余切	atan2	四象限反正切
acot	反余切	asinh	反双曲正弦		
sec	正割	acosh	反双曲余弦		

表 1.2 指数函数

名称	含 义	名称	含 义	名称	含 义
exp	e 为底的指数	log10	10 为底的对数	pow2	2 的幂
log	自然对数	log2	2 为底的对数	sqrt	平方根

表 1.3 复数函数

名称	含 义	名称	含 义	名称	含 义
abs	绝对值	conj	复数共轭	real	复数实部
angle	相角	imag	复数虚部		

表 1.4 取整函数和求余函数

名称	含 义	名称	含 义
ceil	向 $+\infty$ 圆整	rem	求余数
fix	向 0 圆整	round	向靠近整数圆整
floor	向 $-\infty$ 圆整	sign	符号函数
mod	模除求余		

表 1.5 矩阵变换函数

名称	含 义	名称	含 义
fiplr	矩阵左右翻转	diag	产生或提取对角阵
fipud	矩阵上下翻转	tril	产生下三角
fipdim	矩阵特定维翻转	triu	产生上三角
Rot90	矩阵逆时针 90°翻转		

表 1.6　其他函数

名称	含义	名称	含义
min	最小值	max	最大值
mean	平均值	median	中位数
std	标准差	diff	相邻元素的差
sort	排序	length	个数
norm	欧氏(Euclidean)长度	sum	总和
prod	总乘积	dot	内积
cumsum	累计元素总和	cumprod	累计元素总乘积
cross	外积		

Matlab 可以将计算结果以不同精确度的数字格式显示，这只需在指令窗口功能选单上选择 Numerical Format，或者直接在指令窗口中键入以下各个数值显示格式的指令。以 π 值为例，不同精确度的数值格式显示见表 1.7。

表 1.7　数值格式列表

指　令	数　值	说　明
Format short	3.1416	预设的 4 位有效小数位数
Format long	3.14159265358979	14 位有效小数位数
Format short e	3.1416e+000	4 位有效小数位数加上指数表格式

Matlab 对变量的名称有以下规定：

(1) 变量名称的英文大小写是有区别的，如 apple、Apple、AppLe 是三个不同变量。

(2) 变量的长度上限为 19 个字元。

(3) 变量名的第一个字必须是一英文字，随后可以使用英文字符、数字或下画线。

表 1.8 列出了 Matlab 所定义的特别变量及其意义。

表 1.8　Matlab 定义的系统变量

变量名	意　义	变量名	意　义
help	在线帮助，如 hel pquit	pi	内建的 π 值
who	列出所有定义过的变量名称	inf	∞值，无限大 $(\frac{1}{0})$
ans	预设的计算结果的变量名	NaN	无法定义一个数目 $(\frac{0}{0})$
eps	Matlab 定义的正的极小值 2.2204e−16		

Matlab 利用上下两个光标键调出使用过的指令。按下则前一次指令重新出现，之后再按 Enter 键，即再执行前一次的指令；而“上”键的功用则是往后执行指令。

键入 who 可以查看所有定义过的变量名称；而键入 clear 则是去除所有定义

过的变量名称。如果只是要去除 x 及 y 两个变量则可以键入：

```
clear x y
```

Ctrl－C（即同时按 Ctrl 及 C 两个键，下同）可以用来中止执行中的 Matlab 工作。

Matlab 系统中的在线帮助有以下三种方式。

(1) 利用 help 指令。如果已知要找的题材（topic)是什么，则直接键入 help<topic>即可。所以即使身旁没有使用手册，也可以使用 help 指令查询不熟悉的指令或题材之用法，例如：

```
help sqrt,
help topic
```

(2) 利用 lookFor 指令。它可以根据键入的关键字(Key－word)(即使这个关键字并不是 Matlab 的指令)列出所有相关的题材，例如：

```
lookFor cosine,
lookFor sine
```

(3) 利用指令窗口功能选单中的 Help，从中选取 Table of Contents(目录)或 Index(索引)。例如：

```
>>help sqrt
SQRT Square root.
SQRT(X) is the square root of the elements of X. Complex
results are produced if X is not positive.
>>help monkey
monkey not found.
>> * lookFor tangent
ACOT Inverse cotangent.
ACOTH Inverse hyperbolic cotangent.
ATAN Inverse tangent.
ATANH Inverse hyperbolic tangent.
ATAN2 Four quadrant inverse tangent.
COT Cotangent.
COTH Hyperbolic cotangent.
TAN Tangent.
TANH Hyperbolic tangent.
>>help atan
ATAN Inverse tangent.
ATAN(X) is the arctangent of the elements of X. See also ATAN2.
```

Matlab 本身还自带了 Demo 程序(见图 1.2)，它是各个工具包的入门向导，通过图形界面用户可以很容易地掌握 Matlab 的基本用法。

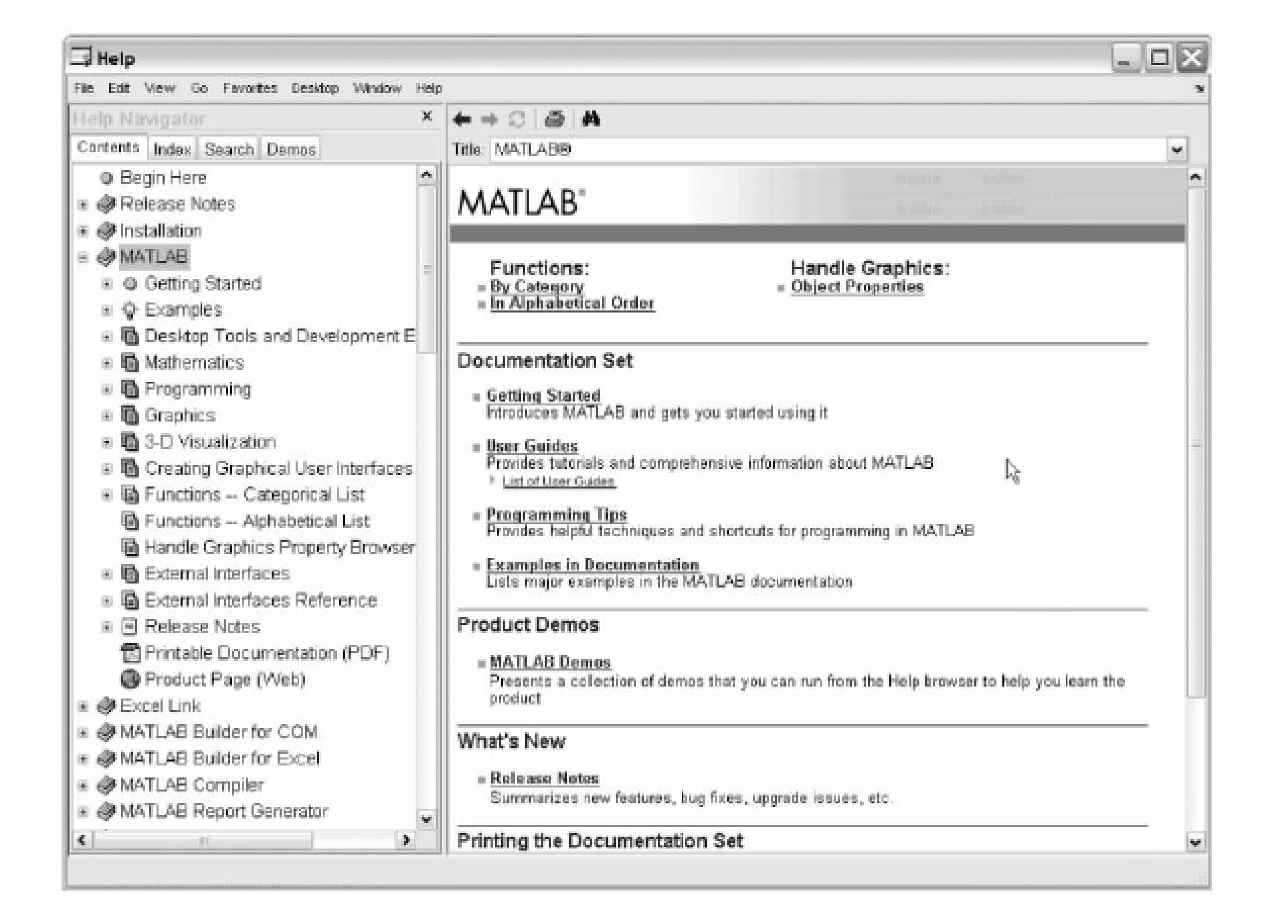


图 1.2　Matlab 自带的 Demo 程序

1.2.2　数组和矩阵

Matlab 的运算事实上是以数组（Array）及矩阵（Matrix）方式在作运算，而这两者在 Matlab 的基本运算性质是不同的，数组强调元素对元素的运算，而矩阵则采用线性代数的运算方式。

声明一个变量为数组或矩阵时，如果是要个别键入元素，需要用中括号将元素置于其中。数组为一维元素所构成，而矩阵为多维元素所组成，例如：

```
>>x=[1 2 3] % 一维 1×3 数组
>>x=[1 2 3; 4 5 6] % 二维 2×3 矩阵，以“;”区隔各列的元素
>>x=[1 2 3 4 5 6]% 二维 2×3 矩阵，各列的元素分两行键入
```

1. 数组的用法

例如要计算：

$$y=\sin(x),\ 0<x<\pi \text{ 且 } x=0,\ 0.2\pi,\ 0.4\pi,\cdots,\pi$$

(1) 如果用数组方式运算，可以使用下面的指令：

```
>>x=[0 0.2 * pi 0.4 * pi 0.6 * pi 0.8 * pi pi] %注意数组内也可作运算
x =
0 0.6283 1.2566 1.8850 2.5133 3.1416
>>y=sin(x)
y =
```

0 0.5878 0.9511 0.9511 0.5878 0.0000

(2) 要找出数组的某个元素或数个元素,可使用以下的指令:

```
>>x(3) % x 的第三个元素
ans=
1.2566
>>y(5) % y 的第五个元素
ans=
0.5878
>>x(1:5) % 列出 x 的第一个到第五个元素
ans=
0 0.6283 1.2566 1.8850 2.5133
>>y(3:-1:1) % 列出 y 的第三个到第一个元素,3 为起始值,1 为终止值,-1 为增量
ans=
0.9511 0.5878 0
>>x(2:2:6) % 列出 x 的第二个到第六个元素,2 为起始值,6 为终止值,2 为增量
ans =
0.6283 1.8850 3.1416
>>y([4 2 5 1]) % 列出 y 的元素,排列元素依次为原来 y 数组的 4、2、5、1 个
ans =
0.9511 0.5878 0.5878 0
```

(3) 如果要建立的数组的元素数目很大,则可以采用以下几种方式:

```
>>x=(0:0.02:1);  %以“:”区隔起始值=0,增量值=0.02,终止值=1
>>x=linspace(0,1,51);%利用 linspace 区隔起始值=0,终止值=1,之间的元素数目=51
>>x=(0:0.01:1)*pi;%注意数组外也可作运算
>>a=1:5, b=1:2:9 %这两种方式更直接
a =
1 2 3 4 5
b =
1 3 5 7 9
>>c=[b a] % 可利用先前建立的数组 a 及数组 b ,组成新数组
c =
1 3 5 7 9 1 2 3 4 5
>>d=[b(1:2:5) 1 0 1] % 由数组 b 的三个元素再加上三个元素组成
d =
1 5 9 1 0 1
```

(4) 数组的算术运算。包括加、减、乘、除(又分为左除和右除)、乘方和转置,

需要注意的是除了加减符号外其余的数组运算符号均须多加“.”符号。下面是一些数组运算的例子：

```
>>a=1:5; a-2 % 从数组 a 减 2
ans =
-1 0 1 2 3
>>2*a-1 % 以 2 乘数组 a 再减 1
ans =
1 3 5 7 9
>>b=1:2:9; a+b % 数组 a 加数组 b
ans =
2 5 8 11 14
>>a.*b % 数组 a 及数组 b 中的元素与元素相乘
ans =
1 6 15 28 45
>>a./b % 数组 a 及数组 b 中的元素与元素相除
ans =
1.0000 0.66667 0.6000 0.5714 0.5556
>>a.^2 % 数组中的各个元素作二次方
ans =
1 4 9 16 25
>>2.^a % 以 2 为底,以数组中的各个元素为次方
ans =
2 4 8 16 32
>>b.^a % 以数组 b 中的各个元素为底,以数组 a 中的各个元素为次方
ans =
1 9 125 2401 59049
>>b=a' % 数组 b 是数组 a 的转置结果
b =
1
2
3
4
5
```

2. 矩阵运算及相关函数

(1) 几个特殊矩阵。zeros 函数是形成元素皆为 0 的矩阵;ones 函数是形成元素皆为 1 的矩阵；eye 则是产生一个单位矩阵,之所以称为 eye,是取其发音与原来单位矩阵符号 ***I*** 相同,而又避免与定义复数中的虚部所用的符号 i 雷同,所以改以 eye 替代。

上述三个函数的使用语法都相似，如 zeros(m)可以产生一个 $m \times m$ 的正方矩阵，而 zeros(m,n)产生的是 $m \times n$ 的矩阵。也可以使用这三个函数将一 $m \times n$ 矩阵原来元素全部取代成 0、1 或单位矩阵的值，不过要加上 size 指令来指出其矩阵大小是 $m \times n$，所以语法为：

zeros(size(A))

其中 **A** 是原来矩阵。

```
>>A=zero(2) % 0 的矩阵
A =
0 0
0 0
>>B=zeros(2,3)
B =
0 0 0
0 0 0
>>C=[1 2; 3 4; 5 6];
>>size(C) % 使用 size 指令得到 C 矩阵的大小
ans =
3 2
>>D=zeros(size(C));% 加上 size 指令将矩阵 C 原来的元素全部以 0 取代
>>A=ones(2), B=ones(2,3) % 1 的矩阵
A =
1 1
1 1
B =
1 1 1
1 1 1
>>C=[1 2; 3 4; 5 6];
>>D=ones(size(C));
>>A=eye(2), B=eye(2,3) % 单位矩阵
A=
1 0
0 1
B=
1 0 0
0 1 0
>>C=[1 2; 3 4; 5 6];
>>D=eye(size(C));
```

(2) 矩阵运算。其所采用的运算符和数组运算基本相似，例如：

```
>>A=[2 5 1; 7 3 8; 4 5 21; 16 13 0];
>>A'% A的转置矩阵
A =
2 7 4 16
5 3 5 13
1 8 21 0
>>A=[4-1 3]; B=[-2 5 2];
>>dot_prod = sum(A.*B) % 两个数组作内积
dot_prod =
-7
>>c=dot(A,B) % 用dot函数也可作内积运算
c =
-7
>>A=[4; -1; 3];
>>dot_prod = sum(A'.*B); % 如果A是行数组则先作转置,再作内积
>>F=[2 5 -1]; G=[0 1 -3];
>>out_prod=F'*G; % 两矩阵作外积
>>A=[2,5,1; 0,3,-1];
>>B=[1,0,2; -1,4,-2; 5,2,1];
>>C=A*B % 矩阵相乘,注意两个矩阵的大小须相容
C =
2 22 -5
-8 10 -7
>>A=[2 1; 4 3];
>>A^2 % 矩阵乘方运算
ans =
4 1
16 9
```

(3) 与矩阵转置相关的函数:rot90, fliplr, flipud。它们的用法及说明如下例所示。

```
>>A=[2 1 0; -2 5 -1; 3 4 6];
>>B=rot90(A) % 将A矩阵逆时针转90°
B=
0-1 6
1 5 4
2-2 3
>>A=[1 2; 4 8; -2 0];
>>B=fliplr(A); % 将A矩阵从左向右翻
```

```
>>C=flipud(A); % 将 A 矩阵从上向下翻
>>B, C
B =
2 1
8 4
0 -2
C =
-2 0
4 8
1 2
```

(4) 函数 reshape 则是用来调整矩阵改形,即在矩阵元素总数不变的情况下,改变其列及行的大小,例如:

```
>>A=[2 5 6 -1; 3 -2 10 0];
>>B=reshape(A,4,2); % 将 A 矩阵改成 4×2 的矩阵
>>C=reshape(A,1,8); % 将 A 矩阵改成 8×1 的矩阵
>>B, C
B =
    2 6
    3 10
    5-1
    -2 0
C =
    2 5
    6 1
    3-2
    10 0
```

(5) 读取矩阵内的特定元素,或将特定元素以其他值取代,可用函数 diag、triu 或 tril。diag 是只保留原矩阵的主对角线(Main Diagonal)上的元素,其余的元素以 0 取代。triu、tril 则是分别产生上三角形及下三角形矩阵,其余的元素也以 0 取代。例如:

```
>>V=[1 2 3];
>>A=diag(V)
A =
1 0 0
0 2 0
0 0 3
>>A=[1:2:7; 3:3:12; 4:-1:1; 1:4]
A =
```

```
1 3 5 7
3 6 9 12
4 3 2 1
1 2 3 4
>>B=triu(A)
B =
1 3 5 7
0 6 9 12
0 0 2 1
0 0 0 4
>>C=triu(A,-1)
C =
1 3 5 7
3 6 9 12
0 3 2 1
0 0 3 4
>>D=triu(A,3)
D =
0 0 0 7
0 0 0 0
0 0 0 0
0 0 0 0
>>B=tril(A)
B =
1 0 0 0
3 6 0 0
4 3 2 0
1 2 3 4
>>C=triu(A,-1)
C =
0 0 0 0
3 0 0 0
4 3 0 0
1 2 3 0
>>D=triu(A,3)
D =
1 3 5 7
3 6 9 12
```

```
4 3 2 1
1 2 3 4
```

1.2.3 程序控制语句

1. For 循环语句

For 循环允许一组命令以固定的和预定的次数重复。For 循环的一般形式是：

```
For 变数＝矩阵
运算式；
end
```

在 For 和 end 语句之间的运算式按数组中的每一列执行一次。在每一次迭代中，x 被指定为数组的下一列，即在第 n 次循环中，x＝array(:, n)。例如：

```
>>n=0:1:10;
>>y=n;
>>For i=1:11
y(i)=sin(n(i));
end
>>y
y =
  Columns 1 through 7
      0 0.8415   0.9093   0.1411   -0.7568   -0.9589   -0.2794
  Columns 8 through 11
0.6570   0.9894   0.4121   -0.5440
```

For 循环内接受任何有效的 Matlab 数组。例如：

```
>>a=1;
>>For i=n
y=sin(n(a))
a=a+1
end
y =0
a =2
y =0.8415
a =3
y =0.9093
a =4
y =0.1411
a =5
y =-0.7568
```

```
a =6
y =-0.9589
a =7
y =-0.2794
a =8
y =0.6570
a =9
y =0.9894
a =10
y =0.4121
a =11
y =-0.5440
a =12
```

Matlab 中的 For 循环语句还可以支持嵌套，例如：

```
>>For i=1:11
For j=1:11
y(i)=sin(n(i));
n(j)=n(j) * 10;
end
end
>>y
y =
  Columns 1 through 7
       0  -0.5064  0.9300  -0.8027  -0.1425  -0.9765  -0.5118
  Columns 8 through 11
    0.8586  -0.9957  0.9917  0.9287
>>n
n =
  1.0e+012 *
  Columns 1 through 7
       0  0.1000  0.2000  0.3000  0.4000  0.5000  0.6000
  Columns 8 through 11
    0.7000  0.8000  0.9000  1.0000
```

在使用 For 循环时需要注意以下两点：

(1) 为了得到最大的速度，在 For 循环(while 循环)被执行之前，应预先分配数组，建议最好先使用 zeros 或 ones 等命令来预先配置所需的内存(即矩阵)大小；

(2) 可以利用 break 命令跳出 For 循环。

2. while 循环语句

while 循环以不定的次数求一组语句的值。while 循环的一般形式是：

```
while 条件式,
  运算式;
end
```

只要在表达式里的所有元素为真，就执行 while 和 end 语句之间的运算式。通常，表达式的求值给出一个标量值，但数组值也同样有效。在数组情况下，所得到数组的所有元素必须都为真。只要条件式成立，运算式就会一再被执行。

```
>>x = zeros(1,6); % x是一个1×6的零矩阵
i = 1;
while i <= 6,
x(i) = 1/i;
i = i+1;
end
>>x
x =
1.0000  0.5000  0.3333  0.2500  0.2000  0.1667
```

同样，while 循环也有以下两点需要注意：

(1) 可以利用 break 命令跳出 while 循环；

(2) while 循环可按需要嵌套。

3. if—else—end 分支语句

(1) 最简单的 if—else—end 结构是：

```
if 条件式
  运算式;
end
```

如果在表达式中的所有元素为真(非零)，那么就执行 if 和 end 语句之间的语句。例如：

```
>>if rand(1)>0.5
disp('Hello World!')
end
Hello World!
```

(2) 如果有两个选择，if—else—end 结构是：

```
if 条件式
  运算式
else
  运算式
end
```

在这里，如果表达式为真，则执行第一组命令；如果表达式为假，则执行第二组

命令。例如：

```
>>if rand(1)>0.5
disp('Hello World!')
else
disp('Hello Matlab!')
end
Hello Matlab!
```

（3）当有多种选择时，可以执行下面的语句：

```
if 条件式
  运算式
elseif 条件式
  运算式
elseif 条件式
  运算式
 ⋮
```

最后的这种形式，只有所碰到的、与第一个真值表达式相关的命令被执行；接下来的关系表达式不检验，跳过其余的 if－else－end 结构。而且，最后的 else 命令可有可无。在了解了如何用 if－else－end 结构来决策之后，就有可能提出一种合理的方法来跳出或中断 For 循环和 while 循环

4．switch－case 语句

Matlab 中一般的 switch－case 语句格式为：

```
switch num
case n1
  command
case n2
  command
case n3
  command
 ⋮
otherwise
  command
end
```

一旦 num 为其中的某个值或字符串时，就执行所对应的指令，没有对应时，则执行 otherwise 后的语句。

注意，Matlab switch－case 语句和 C 语言中的区别：

（1）当开关条件式的值等于条件式 1 时，将执行语句段 1，执行完语句段 1 后将转出开关体，无需像 C 语言那样在下一个 case 语句前加 break 语句，所以本结

构在这点上和 C 语言是不同的。

(2) 当需要在开关条件式满足若干个条件式之一时执行某一程序段,则应该把这样的一些条件式用大括号括起来,中间用逗号分隔。

(3) 当前面枚举的各个条件式均不满足时,则将执行 otherwise 语句后面的语句段,此语句等价于 C 语言中的 default 语句。

(4) 在 case 语句引导的各个条件式中,不要用重复的条件式,否则列在后面的开关通路将永远也不能执行。

(5) 程序的执行结果和各个 case 语句的次序无关。

1.2.4 辅助语句

Matlab 中的辅助语句包括:注释语句、中断语句、暂停语句和回显语句。

Matlab 中的符号注释是由“%”开始,也就是说在“%”之后的任何文字都被视为程序的注释。注释的功能是简要地说明程序的内容,过多的注释在程序中或许没有必要,但是我们写程序时往往用了太少的注释。任何可能产生混淆的地方都应该使用注释,适量的注释可在日后想了解程序时节省一些不必要的时间。例如:

```
>>4 * 4 %这是个例子
ans =
  16
```

中断指令 break 用于中止一个循环语句的执行过程,特别可以利用 break 命令跳出 For 循环和 while 循环。

而暂停语句 pause 的作用是使程序暂时停止运行,直到按下 Enter 键,继续执行程序。而 pause(n)是中断 n 秒后,程序自动继续执行。另外,使用 Ctrl-C 可以用来中止执行中的 Matlab 工作。

回显语句的格式如下:

```
echo on/off
```

这条语句控制是否在屏幕上回显 Matlab 正在执行的语句。系统默认的状态是 echo off,这个功能的实现对调试程序很有帮助。

1.2.5 Matlab 的输入与输出语句

Matlab 的输入语句 input 函数用于接收用户的输入,例如:

```
>>x=input('please input a number:')
please input a number:22
x = 22
```

如果需要输入字符串,则可以使用下面的代码:

```
>>x=input('please input a string:','s')
please input a string:this is a string
x = this is a string
```

Matlab 输出语句包括自由格式 (disp)和格式化输出 (fprintf)两种，下面分别举例：

```
>>disp(23+454-29*4)
361
>>disp([11 22 33; 44 55 66; 77 88 99])
11 22 33
44 55 66
77 88 99
>>disp('this is a string')
this is a string
fprintf('The area is %8.5f\n', area) % 注意输出格式前须有符号"%",
%跳行符号须有符号"\"
The area is 12.56637 % 输出值为 8 位数,含 5 位小数
```

可以看到 Matlab 的格式化输出和 C 语言等高级语言的输出函数是类似的。

1.2.6 变量的保存与装载

用户可以使用 save 来将 Matlab 工作间的变量保存到文件中，以便以后调用这些变量。其使用格式如下：

```
save filename variables
```

而从文件中装载变量的格式如下(其中文件的扩展名为.mat)：

```
load filename variables
```

下面举例说明变量的保存和装载。

```
>>a=1:1:100;
>>t=2323;
>>whos
  Name      Size      Bytes    Class
  a         1×100     800      double array
  t         1×1       8        double array
Grand total is 101 elements using 808 bytes
>>save ok a t
>>a=1;
>>t=2:0.1:3;
>>whos
  Name      Size      Bytes    Class
```

```
    a          1×1      8        double array
    t          1×11     88       double array
Grand total is 12 elements using 96 bytes
>>load ok a t
>>whos
    Name       Size     Bytes    Class
    a          1×100    800      double array
    t          1×1      8        double array
Grand total is 101 elements using 808 bytes
```

读者可以比较一下修改前后的差别。

在 Matlab 中可以使用 fopen 和 fclose 函数对普通的文件实现打开、关闭及处理的功能。例如：

```
fp=fopen(fname,ftype)
st=fclose(fp)
>>fp=fopen('ok.mat','r')
fp=
3
>>st=fclose(fp)
st =
    0
```

文件的读取分为不定格式读取和指定格式读取，不定格式读取的格式如下：

```
a=fread(fp,size)
```

例如，要从文件 fp 中读取数据保存到矩阵 ***a*** 中：

```
>>a=fread(fp)
>>whos
Name        Size          Bytes Class
    a       344×1          2752   double array
    fp      1×1            8      double array
    st      1×1            8      double array
    t       1×1            8      double array
Grand total is 347 elements using 2776 bytes
```

而指定格式读取需要使用下面的语句：

```
a=fscanf(fp,Format,size)
```

例如，需要从句柄 fp 所指定的文件中，按字符串 Format 所指定的数据格式读取数据，最后将其保存到矩阵 ***a*** 中：

```
>>str=fscanf(fp,'%s')
str =
```

''

Matlab 将数据写入文件中的函数是 fprintf，下例是将字符串所指定的数据写入到由 fp 所指定的文件中。

```
>>t=0:0.001:1;
>>fp=fopen('ok.mat','w');
>>fprintf(fp,'%d',t);
>>length(t)
ans =
     1001
>>clear
>>fp=fopen('ok.mat','r');
>>fread(fp);
>>whos
  Name       Size          Bytes  Class
  ans      12989×1        103912  double array
  fp          1×1              8  double array
Grand total is 12990 elements using 103920 bytes
>>length(ans)
ans =
12989
```

读者前后比较一下就可以看出这些函数在使用上的不同之处。

1.3 M 文件与 M 函数

Matlab 程序大致分为两类：M 脚本文件（M－Script，简称为 M 文件）和 M 函数（M－function），它们均是普通的由 ASCII 码构成的文件。

M 文件中包含一组由 Matlab 语言所支持的语句，它类似于 DOS 下的批处理文件，它的执行方式很简单，用户只需在 Matlab 的提示符下键入该 M 文件的文件名，这样 Matlab 就会自动执行该 M 文件中的各条语句，并将结果直接返回到 Matlab 的工作空间。M 函数格式是 Matlab 程序设计的主流。

Matlab 的 M 函数是由 function 语句引导的，其基本格式如下：

```
function [返回变量列表] = 函数名（输入变量列表）
注释说明语句段，由“%”引导
输入变量、返回变量格式的检测
函数体语句
```

这里输入变量和返回变量的实际个数分别由 nargin 和 nargout 两个 Matlab 保留变量来给出，只要进入该函数，Matlab 就将自动生成这两个变量，不论用户是

否直接使用这两个变量。如果返回变量多于一个，则应该用方括号将它们括起来，否则可以省去方括号。输入变量和返回变量之间用逗号分隔。注释语句段的每行语句都应该由“%”引导，“%”后面的内容不执行，只起注释作用。用户采用 help 命令则可以显示出注释语句段的内容。此外，标准的变量个数检测也是必要的，如果输入变量或返回变量格式不正确，则应该给出相应的提示。下面用例子来说明函数编程的格式与方法。

假设现在需要生成一个 $n \times m$ 阶的 Hilbert 矩阵，它的第 i 行第 j 列的元素值为 $1/(i+j-1)$。需要在编写的函数中实现下面几点：

(1) 如果只给出一个输入参数，则会自动生成一个方阵，即令 $m=n$；

(2) 在函数中给出合适的帮助信息，包括基本功能、调用方式和参数说明；

(3) 检测输入变量和返回变量的个数，如果有错误则给出错误信息；

(4) 如果调用时不要求返回变量，则将显示结果矩阵。

根据 Matlab 函数编写格式和上述要求，可以编写出下面的代码：

```
function A=myhilb(n, m)
%MYHILB a demonstrative M-function.
% A=MYHILB(N, M) generates an N by M Hilbert matrix A
% A=MYHILB(N) generates an N by N square Hilbert matrix.
% MYHILB(N,M) displays ONLY the Hilbert matrix, but do not return any
% matrix back to the calling function.
%
%See also: HILB.
% Designed by Professor Dingyu XUE, Northeastern University, PRC
% 5 April, 1995, Last modified by DYX at 21 March, 2000
if nargout>1, error('Too many output arguments.'); end
if nargin==1, m=n;
elseif nargin==0 | nargin>2
error('Wrong number of iutput arguments.');
end
A1=zeros(n,m);
For i=1: n
For j=1:m
A1(i,j)=1/(i+j-1);
end, end
if nargout==1, A=A1; elseif nargout==0, disp(A1); end
```

这样规范编写的函数用 help 命令可以显示出其帮助信息：

```
>>help myhilb
MYHILB a demonstrative M-function.
```

```
A=MYHILB(N, M) generates an N by M Hilbert matrix A.
A=MYHILB(N) generates an N by N square Hilbert matrix.
MYHILB(N,M) displays ONLY the Hilbert matrix, but do not return any
matrix back to the calling function.

See also: HILB.
```

有了函数之后就可以采用下面的各种方法来调用它,并产生出所需的结果。

```
>>A=myhilb(3,4)
A =
1.0000 0.5000 0.3333 0.2500
0.5000 0.3333 0.2500 0.2000
0.3333 0.2500 0.2000 0.1667
>>A=myhilb(4)
A =
1.0000 0.5000 0.3333 0.2500
0.5000 0.3333 0.2500 0.2000
0.3333 0.2500 0.2000 0.1667
0.2500 0.2000 0.1667 0.1429
>>myhilb(4)
1.0000 0.5000 0.3333 0.2500
0.5000 0.3333 0.2500 0.2000
0.3333 0.2500 0.2000 0.1667
0.2500 0.2000 0.1667 0.1429
```

用户在安装 Matlab 的时候会发现 Matlab 包括很多工具箱,放入一个目录中的为某种目的专门编写的一组 Matlab 函数就可以组成一个工具箱。从某种意义上说,任何一个 Matlab 语言的使用者都可以是工具箱的作者。在一个工具箱中,应该有一个名为 Contents.m 的文件,它用来描述工具箱中所有 Matlab 函数的名称和意义。在该文件中第 1 行应该给出该工具箱的名称,在第 2 行中给出该工具箱的版本与修改时间等信息,然后分类地给出该工具箱中各类函数的最基本功能。注意,本文件中所有的语句都应该是注释语句,由"% "引导,空行也应该由"%"引导。

另外,因为 Matlab 是一种解释性语言,所以即使在某个或某些函数中存在语法错误,但如果没执行到该语句时可能就不会发现该错误,这在一个成功的程序设计中是不能容许的。要查出某目录中所有的 M 函数语法错误,首先应该用 cd 命令进入该目录,然后运行 pcode * 命令进行伪代码转换。因为该命令会将 Matlab 函数转换成伪代码,而在转换过程中该程序将自动翻译每一条语句,所以一旦发现有语法错误,将会停止翻译,给出错误信息。改正了该语法错误后,再重新执

行 pcode 命令，直到没有错误为止。这样会保证目录下所有的程序不含有语法错误。

1.4 Matlab 使用时的一些技巧

因为 Matlab 语言是一种解释性语言，所以有时 Matlab 程序的执行速度不是很理想。下面是 Matlab 使用和编程时的一些技巧和经验，读者可以结合实际应用考虑问题，从而能够获得更好的性能。

1.4.1 避免使用循环

首先应该尽量避免使用循环，循环语句及循环体经常被认为是 Matlab 编程的瓶颈问题。改进这样的状况有以下两种方法。

(1) 尽量用向量化的运算来代替循环操作。下面的例子演示了如何将一般的循环结构转换成向量化的语句。考虑下面无穷级数求和问题：

$$I = \sum_{n=1}^{\infty}\left(\frac{1}{2n}+\frac{1}{3n}\right)$$

如果只求出其中前有限项，如前 100000 项之和（要精确地求出级数的和，无需求 100000 项，几十项往往就能得出满意的精度，这里主要是为了演示循环运算向量化的优越性），则可以采用下面的常规语句进行计算：

```
>> tic, s=0;
For i=1:100000, s=s+(1/2^i+1/3^i); end, s,toc
s =
1.5000
elapsed_time =
1.9700
```

如果采用向量化的方法，则可以得出下面结果：

```
>>tic, i=1:100000; s=sum(1./2.^i+1./3.^i), toc
s =
1.5000
elapsed_time =
0.3800
```

可以看出，采取向量化的方法比常规循环运算效率高得多。

(2) 在必须使用多重循环的情况下，如果两个循环执行的次数不同，则建议在循环的外环执行循环次数少的，内环执行循环次数多的。这样也可以显著提高速度。

考虑生成一个 5×10000 的 Hilbert 长方矩阵，该矩阵的定义是其第 i 行第 j 列元素为 $H\{i,j\}=1/(i+j-1)$。可以由下面语句比较先进行“i=1:5”的循环和

后进行该循环的耗时区别，其效果和前面分析的是一致的。

```
>>tic
For i=1:5
For j=1:10000
H(i,j)=1/(i+j-1);
end
end
toc
elapsed_time =
8.6800
>>tic,
For j=1:10000
For i=1:5
J(i,j)=1/(i+j-1);
end
end
toc
elapsed_time =
25.7000
```

1.4.2 大型矩阵维度的预先确定

给大型矩阵动态地确定维度是一个相当费时间的操作，建议在定义大矩阵时，首先用 Matlab 的内在函数，如 zeros() 或 ones() 对之先进行定维，然后再进行赋值处理，这样会显著减少所需的时间。

再考虑上面的问题，如果输入下面的命令：

```
>>tic
H=zeros(5,10000);
For i=1:5
For j=1:10000
H(i,j)=1/(i+j-1);
end
end
toc
elapsed_time =
1.0400
```

如果采用预先定维的方法，再结合向量化的方法，可以给出下面的 Matlab 语句：

```
>>tic
H=zeros(5,10000);
For i=1:5
H(i,:)=1./[i:i+9999];
end
toc
elapsed_time =
0.060
```

可见，预先定维后，所需要的时间显著地减少了。可以看出，同样一个问题，由于采用了有效的措施，所需的时间就可以从 25.7s 减少到 0.06s，亦即效率提高了 428 倍。

对二重循环这样的特殊问题，还可以使用 meshgrid() 函数构造两个 5×10000 矩阵 ***i*** 和 ***j***，从而直接得出矩阵 ***H***，更进一步地加快速度。例如：

```
>>tic, [i,j]=meshgrid(1:5,1:10000); H=1./(i+j-1); toc
elapsed_time =
0
```

另外还有一些经验，例如，矩阵运算应该尽量采用 Matlab 的内在函数，因为内在函数是由更底层的编程语言 C 构造的，其执行速度显然快于使用循环的矩阵运算。还有，采用更有效的算法，在实际应用中，解决同样的数学问题经常有各种各样的算法，例如，求解定积分的数值解法在 Matlab 中就提供了两个函数：quad() 和 quad8()，其中后一个算法在精度、速度上都明显高于前一种方法。所以说，在科学计算领域是存在“多快好省”的途径的，如果一个方法不能满足要求，可以尝试其他的方法。

最后还可以结合 Mex 文件来解决实际问题，因为在某些情况下，虽然采用了很多措施，但执行速度仍然很慢，比如，耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 语言或 Fortran 语言，按照 Mex 技术要求的格式编写相应部分的程序，然后通过编译连接，形成在 Matlab 可以直接调用的动态连接库（DLL）文件，这样可以显著地加快运算速度。

第 2 章　Matlab 二维图形绘制

Matlab 不但擅长与矩阵相关的数值运算，也适合用在各种科学可视化的处理(Scientific Visualization)，本章将结合实例介绍 Matlab 基本二维平面的各项绘图指令。

2.1　基本绘图指令

在 Matlab 中 plot 是绘制一维曲线的基本函数，使用此函数之前需先定义曲线上每一点的 x 及 y 坐标。下例可画出一条正弦曲线：

```
close all;
x=linspace(0, 2*pi, 100); % 100 个点的 x 坐标
y=sin(x); % 对应的 y 坐标
plot(x,y);
```

这段程序在 Matlab 中执行后的显示结果如图 2.1 所示。

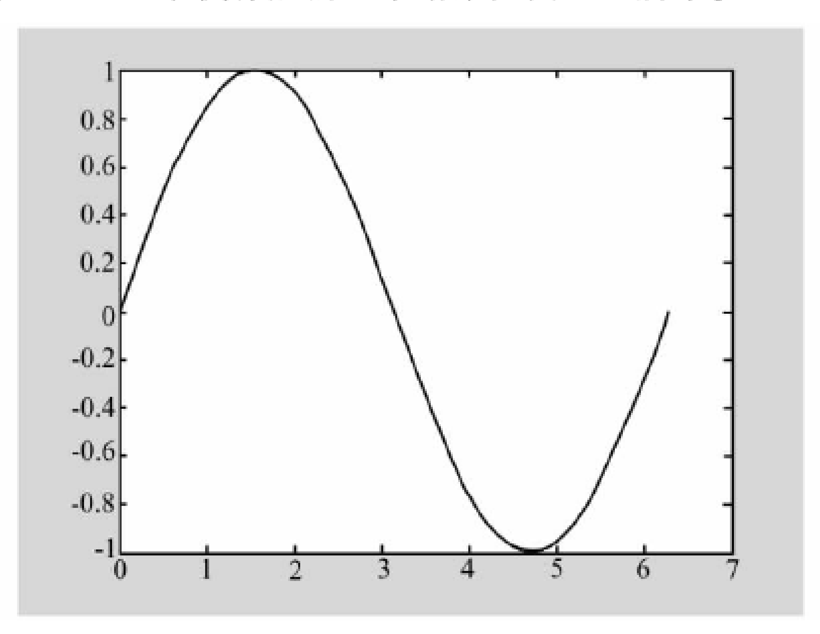


图 2.1　用 plot 函数绘制正弦曲线

图 2.1 中 x 轴、y 轴均使用的是线性刻度，实际上还可以使用对数刻度，表 2.1 所列的是与 plot 类似的几个基本绘图函数，这些函数的使用方法与 plot 相同，但是输出的坐标轴的形式有所区别。

表 2.1 Matlab 提供的基本绘图函数

plot	x 轴和 y 轴均为线性刻度(Linear Scale)	semilogx	x 轴为对数刻度,y 轴为线性刻度
loglog	x 轴和 y 轴均为对数刻度(Logarithmic Scale)	semilogy	x 轴为线性刻度,y 轴为对数刻度

plot 函数还支持多条曲线的同时显示,当需要在一个坐标轴上画出多条曲线时,只需将坐标对依次放入 plot 函数即可,例如:

```
plot(x,sin(x), 'co', x, cos(x), 'g*');
```

上面这条指令在一个坐标轴内同时绘制了正弦曲线和余弦曲线,并使用不同的样式将两者区分开,显示结果如图 2.2 所示。

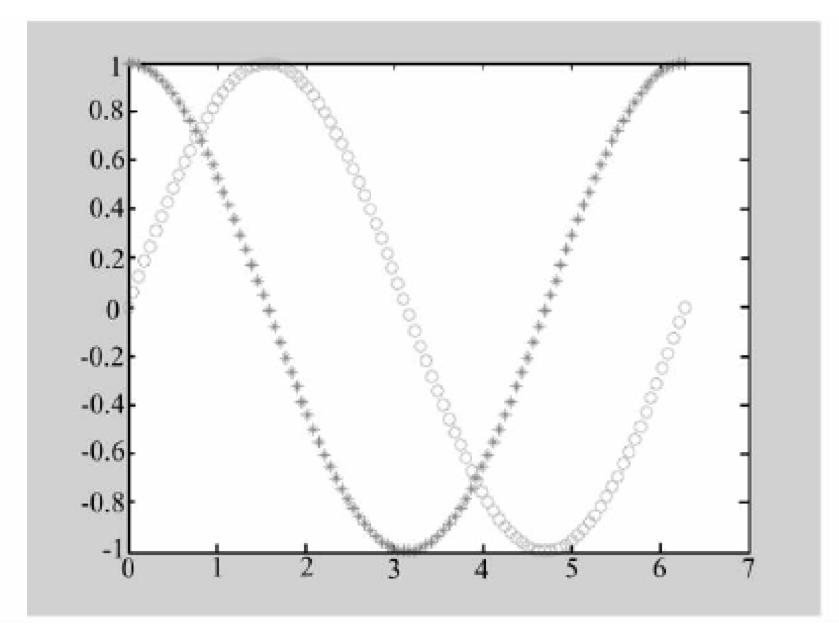


图 2.2 用 plot 函数在同一坐标轴内绘制多条曲线

从图 2.2 可以看到,正弦函数图像的线条颜色是青色,而组成曲线的点是小圆点,这就是使用曲线选项“co”的结果;同样,余弦函数的控制选项是“g *”,表示线条颜色是绿色,而组成线条的点是星号。Matleb一维线条控制选项见表 2.2。

表 2.2 Matlab一维线条控制选项

控制选项	颜色	控制选项	图线型态	控制选项	颜色	控制选项	图线型态
y	黄色	.	点	r	红色	—	实线
k	黑色	o	圆	c	亮青色	:	点线
w	白色	x	×	m	锰紫色	—.	点虚线
b	蓝色	+	+			——	虚线
g	绿色	*	*				

在图形绘制完成后可用 axis([xmin,xmax,ymin,ymax])函数来调整图轴的范围,例如:

```
axis([0, 5, -0.8, 0.8]);
```

这样上面的两个函数就限制在横坐标为 0～5,纵坐标为－0.8～0.8 之间了，而其他部分则不显示出来。

此外,Matlab 也可对图形加上各种注解与处理,例如：

```
xlabel('Input Value'); % x轴注解
ylabel('Function Value'); % y轴注解
title('Two Trigonometric Functions'); % 图形标题
legend('y = sin(x)','y = cos(x)'); % 图形注解
grid on; % 显示格线
```

这个例子中使用 xlabel、ylabel 为坐标轴添加说明文字,另外 title 函数可以为图形加上标题,而 legend 函数的作用是图形的题注,最后 grid 函数为图形加上网格。其显示结果如图 2.3 所示。

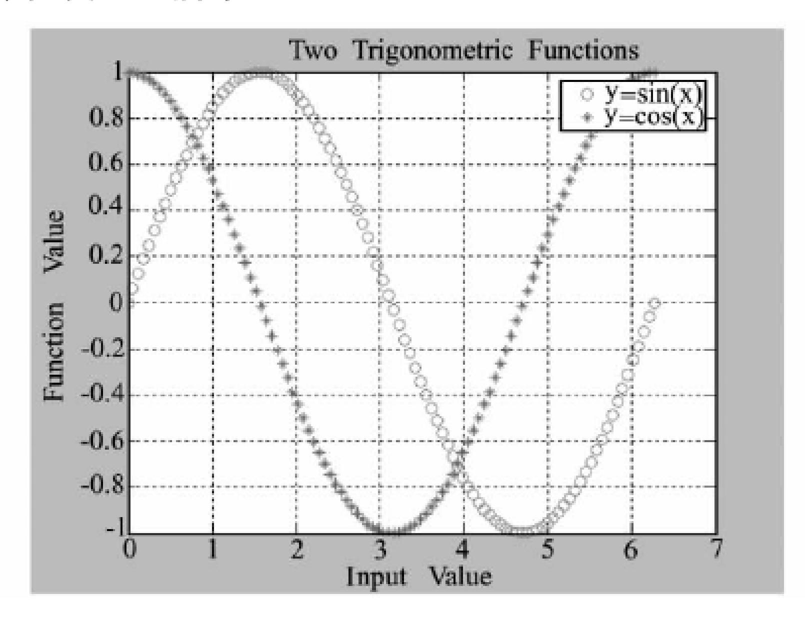


图 2.3　坐标轴的题注、说明和网格

Matlab 还允许将图形窗口划分为几个部分,每一部分分别显示不同的图像,这个功能可用 subplot 函数来实现。这个函数的用法如下：

```
subplot(m,n,p)
```

其中前两个参数的意义是将图形窗口划分为 $m\times n$ 块,第三个参数 p 表示图像的序号。下面是这个函数最简单的用法举例,它将图形窗口划分为四个部分,每一部分显示不同的三角函数。

```
subplot(2,2,1); plot(x, sin(x));
subplot(2,2,2); plot(x, cos(x));
subplot(2,2,3); plot(x, sinh(x));
subplot(2,2,4); plot(x, cosh(x));
```

这段程序在 Matlab 中执行后的显示结果如图 2.4 所示。

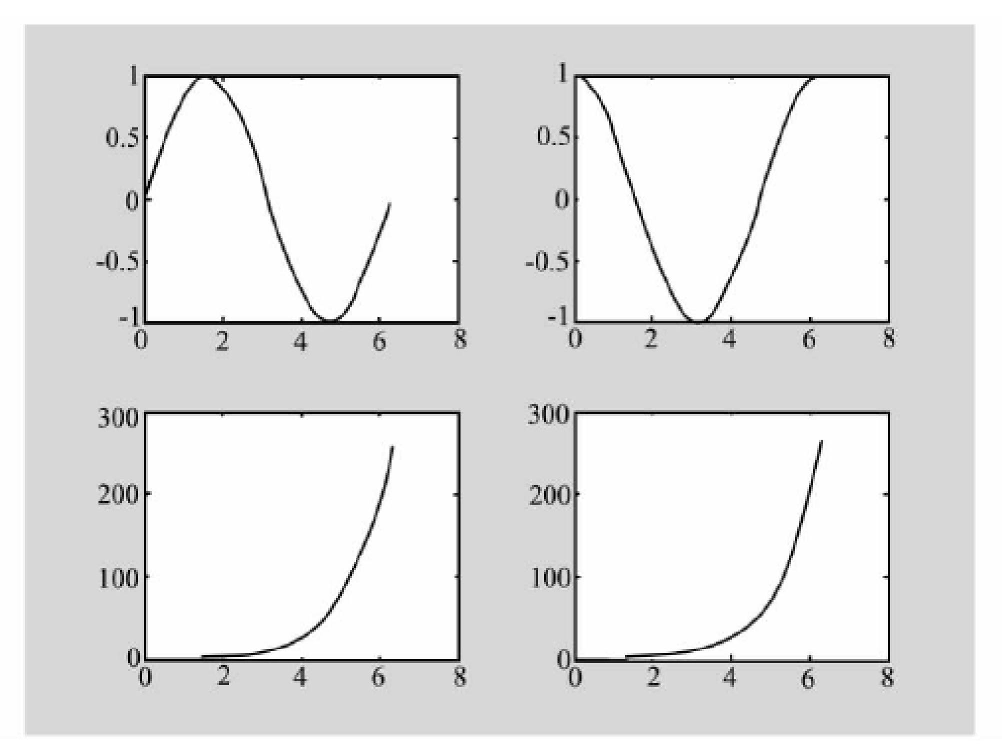


图 2.4 用 subplot 函数划分图形窗口

2.2 绘图选项

上一节介绍了一些 Matlab 中最基本的绘图函数，本节将讲解如何结合绘图选项来控制图形的一些特性。

2.2.1 横轴和纵轴的控制

在 Matlab 中如果需要控制绘图的横轴及纵轴比例，可以使用 axis 配合表 2.3 所列的相关选项。

表 2.3 axis 函数的选项

axis([xmin xmax ymin ymax])	以 xmin、xmax 设定横轴的下限及上限，以 ymin、ymax 设定纵轴的下限及上限
axis auto	横轴及纵轴依照数据大小的上下限来确定，横轴及纵轴的比例是 4:3
axis square	横轴及纵轴的比例是 1:1
axis equal	将横轴及纵轴的尺度比例设成相同值
axis xy	预设值使用卡氏坐标，即将图形原点设在左下角，横轴由左往右递增，纵轴由下往上递增
axis ij	使用矩阵格式，即将图形原点设在左上角，横轴不变，纵轴由上往下递增
axis normal	以预设值画纵轴及横轴

（续）

axis off	将纵轴及横轴取消
axis on	恢复纵轴及横轴

上述各个指令的语法也可以将关键字放在括弧内的单引号之间，如 axis(' ')。下面是应用 axis 的一个例子。

```
x=linspace(0,2*pi,30);
y=sin(x);
z=cos(x);
plot(x,y,x,z)
axis off
axis on
axis('square','equal')
axis('xy','normal')
```

这段程序在 Matlab 中执行后的显示结果如图 2.5 所示。

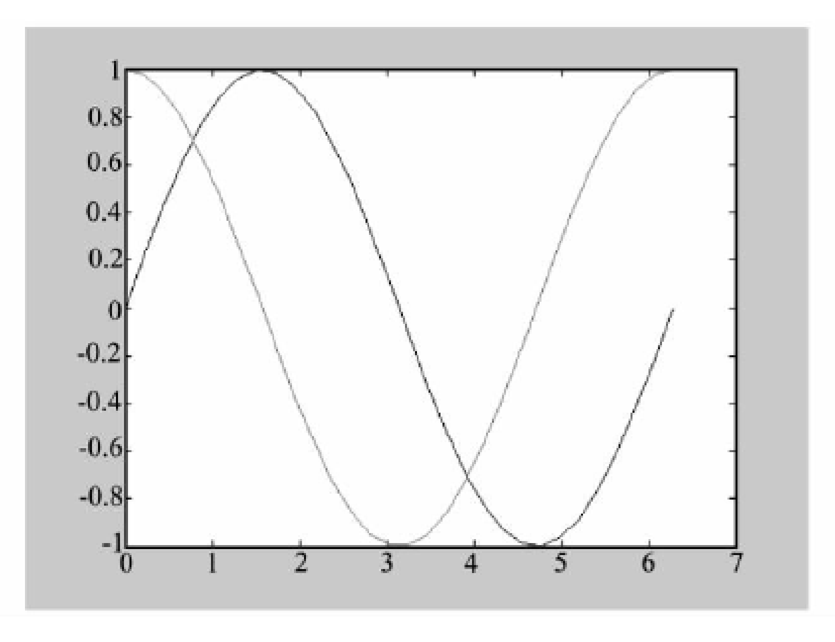


图 2.5　对横纵坐标的控制

2.2.2　图形的放缩

在 Matlab 中，zoom 函数可以将图形放大或缩小，若要将图形放大时用 zoom on 或 zoom out，当不再需要放大或缩小图形时用 zoom off。例如：

```
M=peaks(25);
plot(M)
zoom on
zoom out
zoom off
```

peaks 是 Matlab 内建的一个曲面函数，25 是这个函数矩阵的大小，如果数值越大则画出的曲面越平滑。使用 zoom on 开始放大图形，每按一次 Enter 键图形就放大一次；而 zoom out 是开始缩小图形，每按一次 Enter 键图形就缩小一次；最后使用 zoom off 停止图形放大和缩小功能。这段程序在 Matlab 中执行后的显示结果如图 2.6 所示。

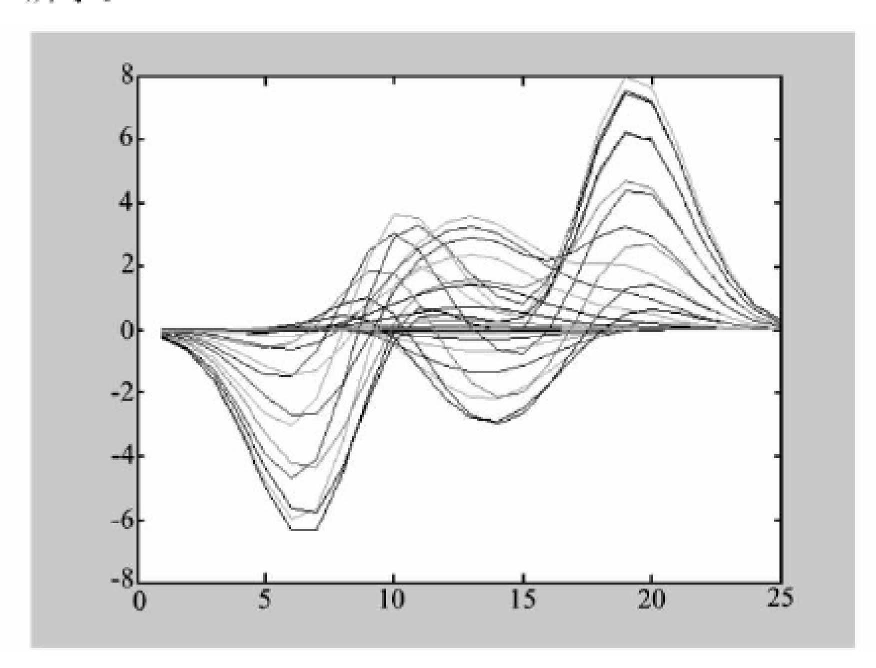


图 2.6　图形的缩放函数 zoom

2.2.3　函数分布的快速绘图

fplot 指令可以用来自动画一个已定义的函数分布图，而无需产生绘图所需要的一组数据作为参数。其语法为：

```
fplot('fun',[xmin xmax ymin ymax])
```

其中 fun 为定义好的函数名称，如 sin、cos 等；而 xmin、xmax、ymin、ymax 则是设定绘图横轴及纵轴的下限及上限。

以下的例子是将函数 $f(x)=\sin(x)/x$ 在 $x\in[-20,20]$，$y\in[0.4,1.2]$的范围内画出：

```
fplot('sin(x)./x',[-20 20 -0.4 1.2]);
title('Fplot of f(x)=sin(x)/x');
xlabel('x'), ylabel('f(x)')
```

这段程序在 Matlab 中执行后的显示结果如图 2.7 所示。

2.2.4　打印和其他选项

Matlab 的指令 print 有两个功能，它可以直接打印所画的图，也可以将所画的图以指定的图形格式存储起来。如果是前者，则可直接使用图形窗口中的打印按

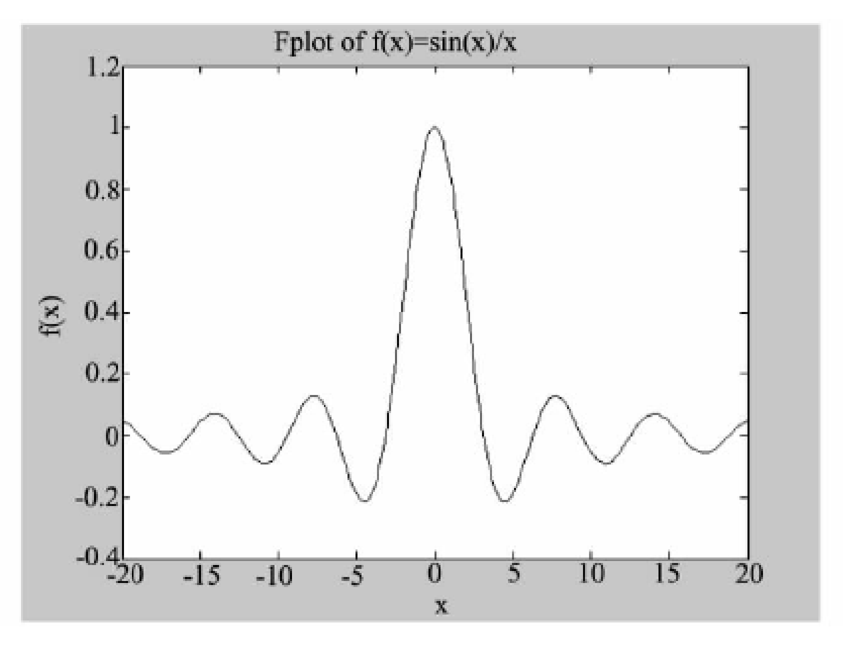


图 2.7 函数分布的快速绘图

钮。以下的指令则是针对后者的方式,其基本语法为:

```
print[filename][－device][－options]
```

其中[ ]代表附加的选项,如果指令是 print,则表示打印当前图形窗口的图形;如果是 print filename,则表示将当前图形窗口的图形存为名为 filename 的文件。打印机的类型由－device 决定,Matlab 直接支持的打印机类型见表 2.4。

表 2.4 Matlab 直接支持的打印机类型

device	说 明	device	说 明
－dps	PostScript 打印机,扩展名为 .ps	－deps	Encapsulated PostScript 打印机,扩展名为 .eps
－dps2	PostScript II 打印机,扩展名为 .ps	－deps2	Encapsulated PostScript II 打印机,扩展名为 .eps

如果需要在所画图中的曲线的某处加上符号,而又需要随意放置这些符号,则可以使用指令 ginput ,它允许以鼠标或键盘上下左右键在屏幕上输入要加上符号的坐标。下面的例子是一个有 8 个峰顶及峰谷的函数分布图($y=\sin(x)/x$),以鼠标方式将符号加在这些峰值上,突出显示这些极值,其语法为:

```
[x,y]=ginput(n)。
```

下面是例子的代码:

```
x=linspace(－2*pi,2*pi,60);
y=sin(x).^2./(x+eps); % 注意,加上 eps 可避免当 x 趋近零时,y 会无法定义
plot(x,y)
[a,b]=ginput(8); % 依次在屏幕上输入 8 点的坐标
hold on
plot(a,b,'co') % 根据输入的坐标值将符号画在图上适当位置
hold off
```

这段程序在 Matlab 中执行后的显示结果如图 2.8 所示。

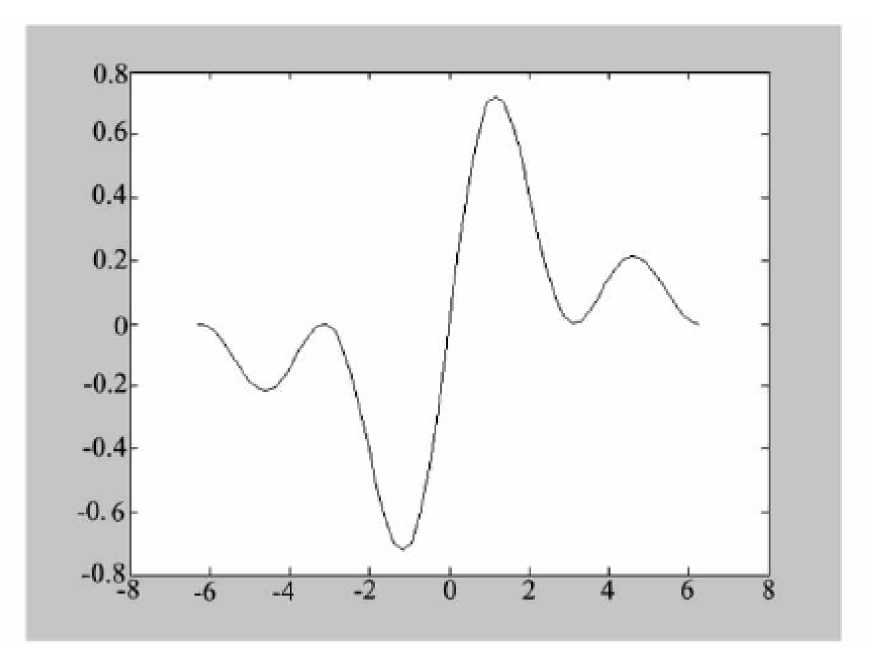


图 2.8　指令 ginput 示例

2.3　特殊的二维图形的绘制

除了标准的 plot() 函数外，为了绘制特殊的二维图形，Matlab 还提供了一些其他函数，具体调用格式和意义见表 2.5。

表 2.5　特殊的二维图形绘制函数

名　称	含　义	调用格式
bar	长条图	bar(x, y)
comet	建立彗星流动图	comet(x, y)
errorbar	图形加上误差范围	errorbar(x, y, l, u)
fplot	较精确的函数图形	fplot(x, y)
polar	极坐标图	polar(x, y)
hist	累计图	hist(y, n)
rose	极坐标累计图	rose(theta, x)
stairs	阶梯图	stairs(x, y)
stem	针状图	stem(x, y)
fill	实心图	fill(x, y, c)
feather	羽毛图	feather(x, y)
compass	罗盘图	compass(x, y)
quiver	向量场图	quiver(x, y)

下面是绘制彗星状轨迹绘制的例子。设函数：

$$f(x)=\tan(\sin(x))-\sin(\tan(x))$$

选定自变量 x 的变化范围为 $x\in[-p,p]$，则可以由下面的函数绘制出不同模式的图形：

```
x=-pi:pi/100:pi;
y=tan(sin(x))-sin(tan(x));
comet(x,y);
```

这段程序在 Matlab 中执行后的显示结果如图 2.9 所示。

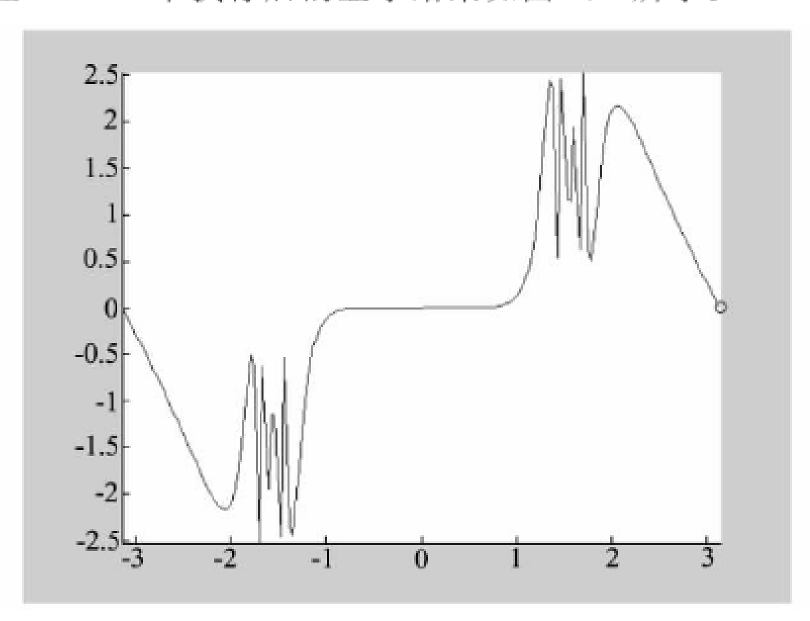


图 2.9　绘制彗星状轨迹

绘制极坐标曲线用 polar(***r***,***t***) 函数，其中 ***r*** 为幅值向量，***t*** 为角度向量。例如，绘制函数 $r=\cos(5q/4)+1/3$；其中 $q\in[0,8p]$，绘制极坐标曲线。

```
t=0:.1:8*pi;
r=cos(5*t/4)+1/3;
polar(t,r)
```

这段程序在 Matlab 中执行后的显示结果如图 2.10 所示。

利用下面的 Matlab 提供的绘图命令还可以绘制出其他一些二维曲线。例如：

```
x=-2:0.1:2; y=sin(x);
subplot(221);
feather(x,y); xlabel('(a) feather()')
subplot(222);
stairs(x,y); xlabel('(b) stairs()')
subplot(223);
stem(x,y); xlabel('(c) stem()')
subplot(224);
```

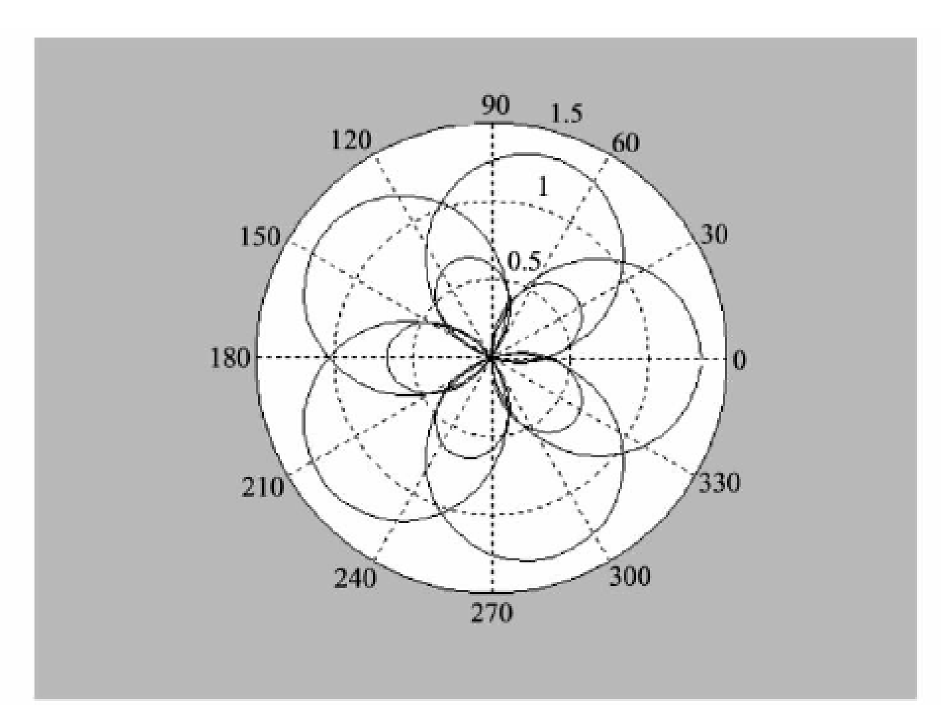


图 2.10　绘制极坐标曲线

```
fill(x,y,'r'); xlabel('(d) fill()')
```

这段程序在 Matlab 中执行后的显示结果如图 2.11 所示。

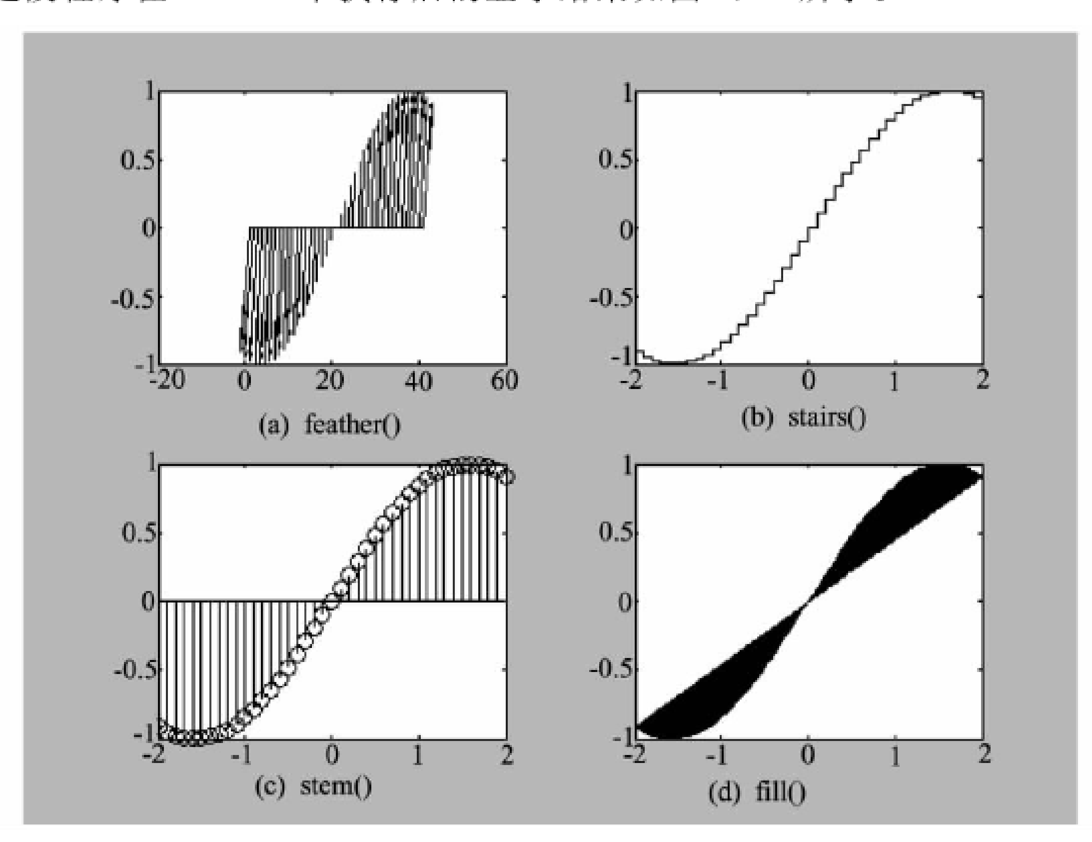


图 2.11　feather、stairs、stem、fill 函数示例

最后是另外一组特殊二维图形的图像举例，其代码如下：

```
t=-10:1:10;
subplot(2,2,1);
```

```
bar(t,cos(t));
subplot(2,2,2);
compass(t,cos(t));
subplot(2,2,3);
rose(t,cos(t));
subplot(2,2,4);
fill(t,cos(t),'b');
```

这段程序在 Matlab 中执行后的显示结果如图 2.12 所示。

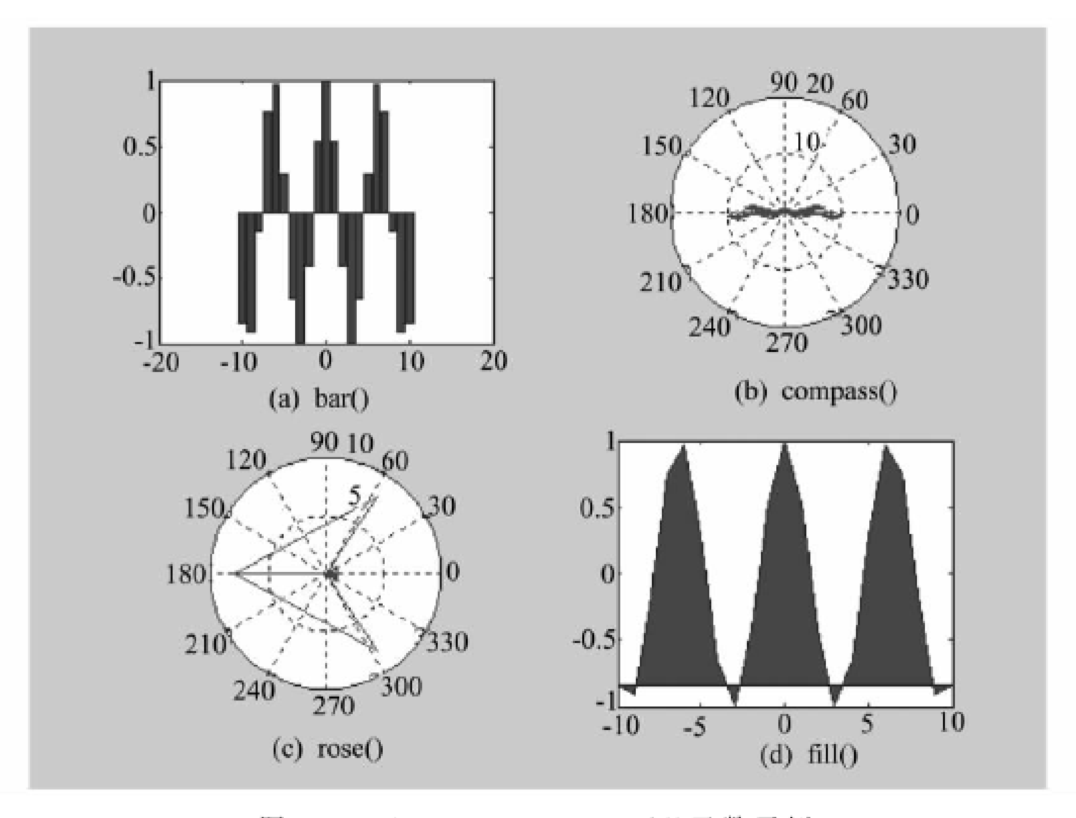


图 2.12 bar、compass、rose、fill 函数示例

第3章　Matlab 三维图形及动画的绘制

3.1　三维图形的绘制

3.1.1　基本的三维图形绘制函数

类似于二维曲线，三维曲线的标准绘制函数为 plot3。plot3 函数将绘制二维图形的函数 plot 的特性扩展到三维空间。函数格式除了包括第三维的信息（如 z 方向）之外，与二维函数 plot 相同。plot3 一般语法调用格式是 plot3(**x**1，**y**1，**z**1，**S**1，**x**2，**y**2，**z**2，**S**2，…)，这里 **x**n，**y**n 和 **z**n 是向量或矩阵，**S**n 是可选的字符串，用来指定颜色、标记符号等。

下面是一个三维螺线的例子：

```
t=0; pi/50; 10*pi;
x=sin(t); y=cos(t); z=t;
h=plot3(x, y, z)
set(h,'LineWidth',4*get(h,'LineWidth'));
grid
```

这条螺线以 z 为自变量，x、y 分别是 z 的正弦函数和余弦函数，而自变量以 $\pi/50$ 为步长，从 0 到 10π，这里的 h 是一个 Line 类型对象，由 plot3 函数创建。这段程序在 Matlab 中执行后的显示结果如图 3.1 所示。

上面这个例子使用 grid 函数来显示网格，如果需要改变网格的属性，可以使用 set 指令来设置：

```
set(axes_handle,'XGrid','on')
```

下面是另一个三维螺线的例子：

```
t=(0:0.1:3)*pi;
x=tan(t);
y=cos(t);
z=sin(t);
plot3(x,y,z,'bo--');
```

这段程序在 Matlab 中执行后的显示结果如图 3.2 所示。

这个例子将 x、y、z 分别赋值为 tan 函数、cos 函数和 sin 函数，调用 plot3 指令时指定了线条的样式，其中 b 代表线条的颜色是蓝色，o 代表结点以圆圈的形式标

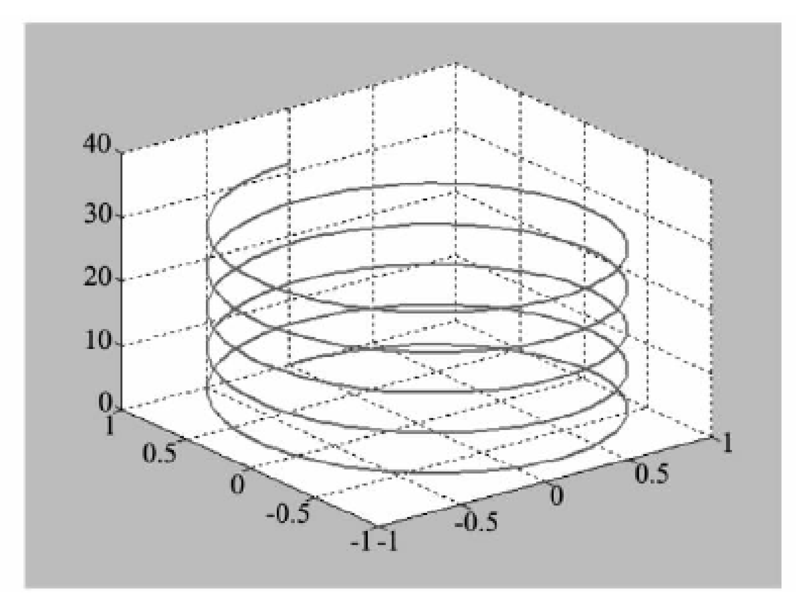


图 3.1　用 plot3 函数绘制三维螺线

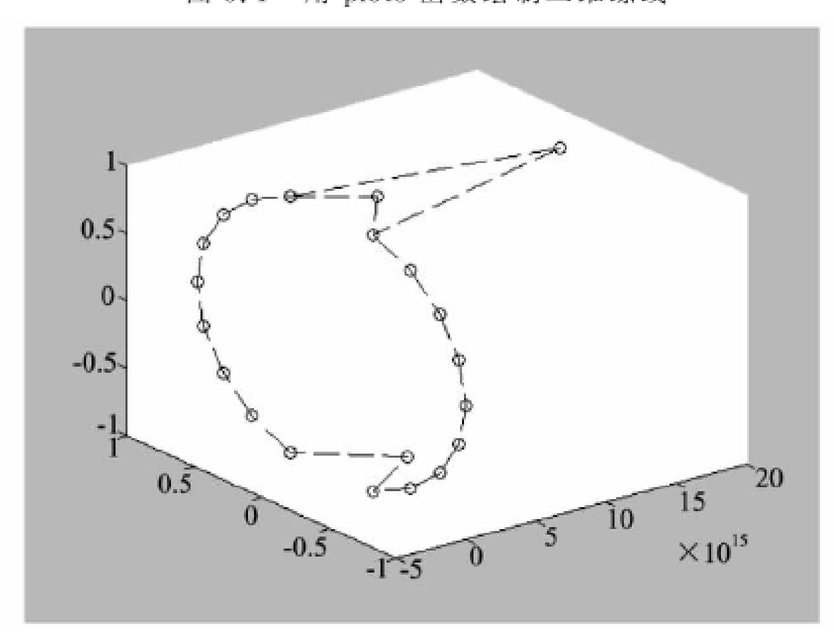


图 3.2　用 plot3 函数绘制三维螺线的另一个例子

记出来，"——"代表以虚线的形式绘制曲线。plot3 函数中线条的样式与二维曲线绘制的样式完全一样，这里不再重复。

plot3 还可以重叠显示多条直线或曲线。例如，增加维数的 plot3 命令可以使多个二维图形沿一个轴排列起来，而不是直接将二维图形叠到另一个上面。下面是一个重叠显示的例子：

```
x=linspace(0,3*pi);
z1=sin(x);
z2=sin(2*x);
z3=sin(3*x);
y1=zeros(size(x));
y3=zeros(size(x));
y2=y3/2;
```

```
plot3(x, y1, z1, x, y2, z2, x, y3, z3);
grid;
xlabel('x-axis');
ylabel('y-axis');
zlabel('z-axis');
title('sin(x),sin(2x),sin(3x)')
```

这个例子首先指定了 x 的范围，然后将 $z1$、$z2$、$z3$ 分别赋值为 sin(x)、sin(2x)、sin(3x)，然后将三条正弦函数同时显示出来，最后指定三个坐标轴的名称并使用 grid 函数画网格，这段程序在 Matlab 中执行后的显示结果如图 3.3 所示。

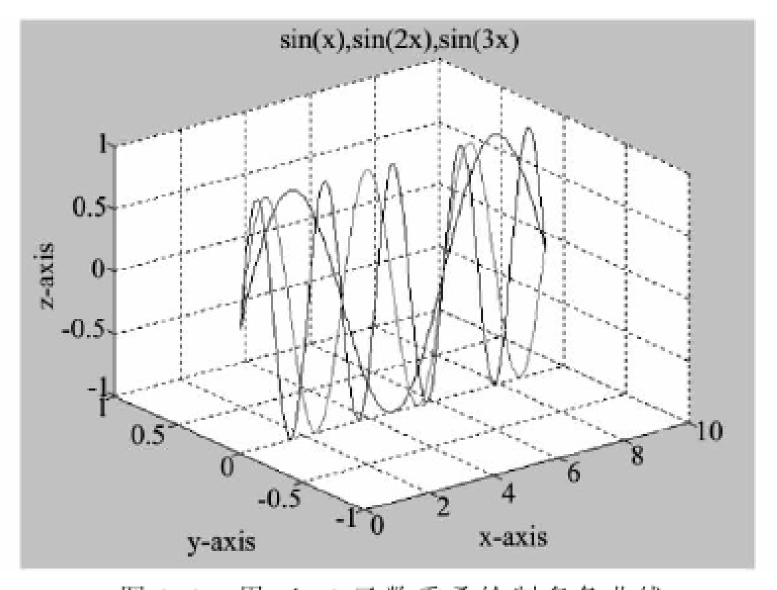


图 3.3　用 plot3 函数重叠绘制多条曲线

也可以沿着其他坐标轴排列，例如，将上面的 plot3 指令改为：

```
plot3(x, z1, y1, x, z2, y2, x, z3, y3)
```

调换 z 和 y 的顺序后，显示结果如图 3.4 所示。

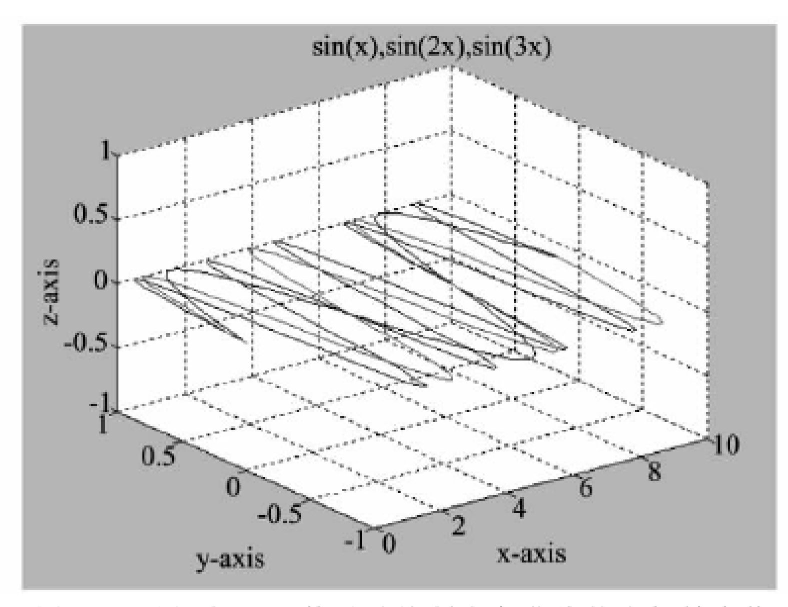


图 3.4　用 plot3 函数重叠绘制多条曲线的坐标轴变换

3.1.2 三维网格图

Matlab 利用在 $x-y$ 平面的矩形网格点上的 z 轴坐标值定义了一个网格曲面。Matlab 通过将相邻的点用直线连接起来形成网状曲面，例如，用 Matlab 的函数 peaks 可以画一个简单的曲面。在下面的例子中可以看到 peaks 函数产生的曲面包含了三个局部极大点和三个局部极小点。

```
[x,y,z]=peaks(30);
mesh(x,y,z);
grid;
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
title('MESH of PEAKS')
```

这段程序在 Matlab 中执行后的显示结果如图 3.5 所示。

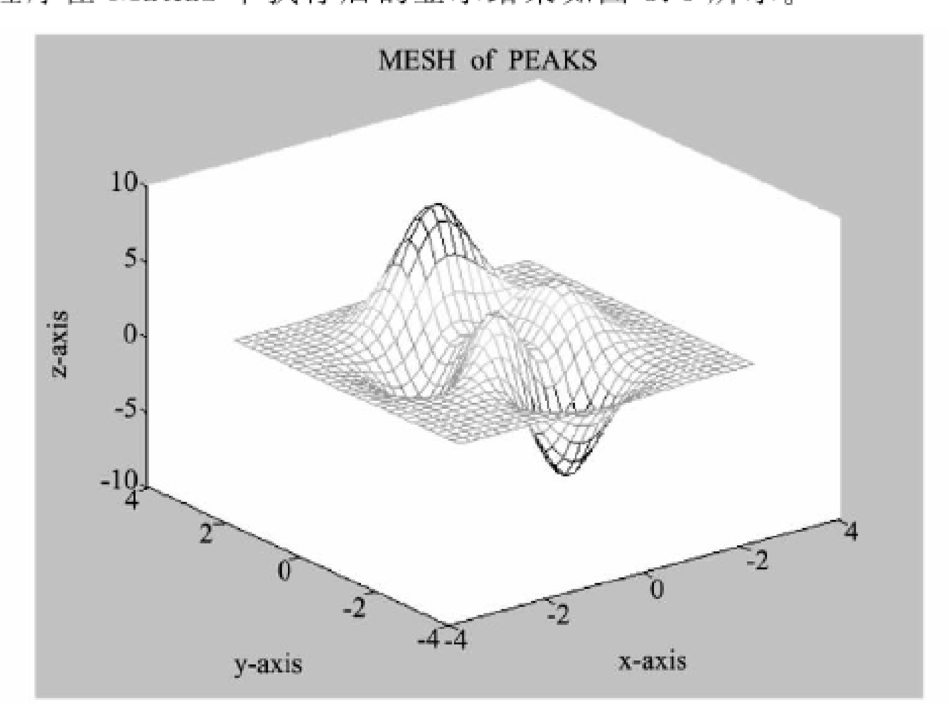


图 3.5　三维网格绘制函数 mesh 示例

采用 mesh 函数的变化形式 meshz、meshc 等可以产生不同的效果，下面的例子中先使用 subplot 函数将整个三维坐标轴划分为四个部分，然后分别以四种形式绘制同一个 peaks 函数。

```
[x,y,z] =peaks;
subplot(2,2,1);
meshz(x,y,z);
axis([-inf inf -inf inf -inf inf]);
subplot(2,2,2);
waterfall(x,y,z);
axis([-inf inf -inf inf -inf inf]);
```

```
subplot(2,2,3);
meshc(x,y,z);
axis([-inf inf -inf inf -inf inf]);
subplot(2,2,4);
surfc(x,y,z);
axis([-inf inf -inf inf -inf inf]);
```

这段程序在 Matlab 中执行后的显示结果如图 3.6 所示。

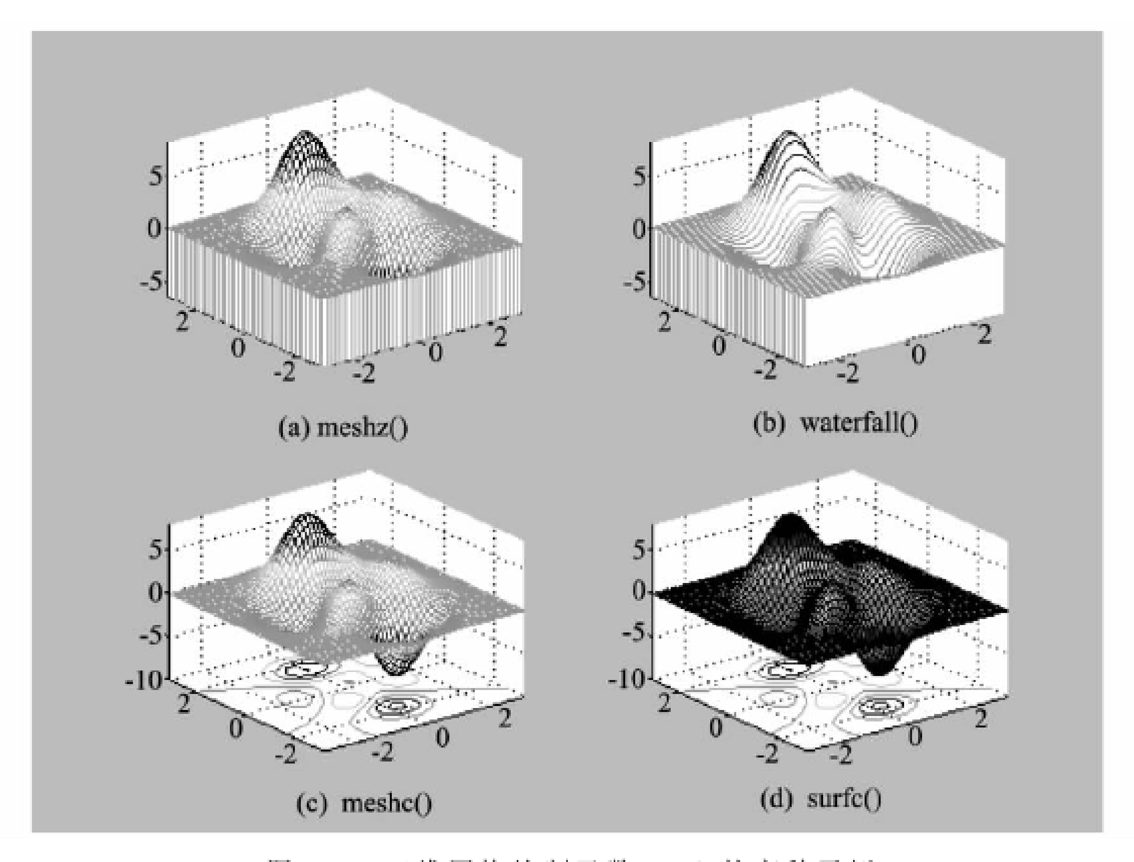


图 3.6　三维网格绘制函数 mesh 的变种示例

其中 meshz 给曲面加上围裙效果，waterfall 在 x 轴方向产生水流效果，meshc 同时画出网状图与等高线，surfc 同时画出曲面图与等高线。surfc 是绘制的对象是三维曲面图，这个函数将在后面的内容中介绍。

mesh 函数还可以和 meshgrid 函数联合使用，如下例中，x 和 y 由 meshgrid 函数生成，然后计算 z，最后将三个向量参数传递给 mesh 函数。

```
Z=f(x,y)=(x*x-2*x)e^(-x*x-y*y-x*y)
```

首先可以调用 meshgrid 函数生成 $x-y$ 平面的网格。即该函数可以产生一个网格分割，其横坐标从−3 到 3，步长为 0.1；纵坐标从−2 到 2，步长为 0.1。然后由公式计算出曲面的矩阵 $\boldsymbol{z}$。最后调用 mesh 函数绘制曲面的三维表面网格图形。

```
[x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
mesh(x,y,z)
```

这段程序在 Matlab 中执行后的显示结果如图 3.7 所示。

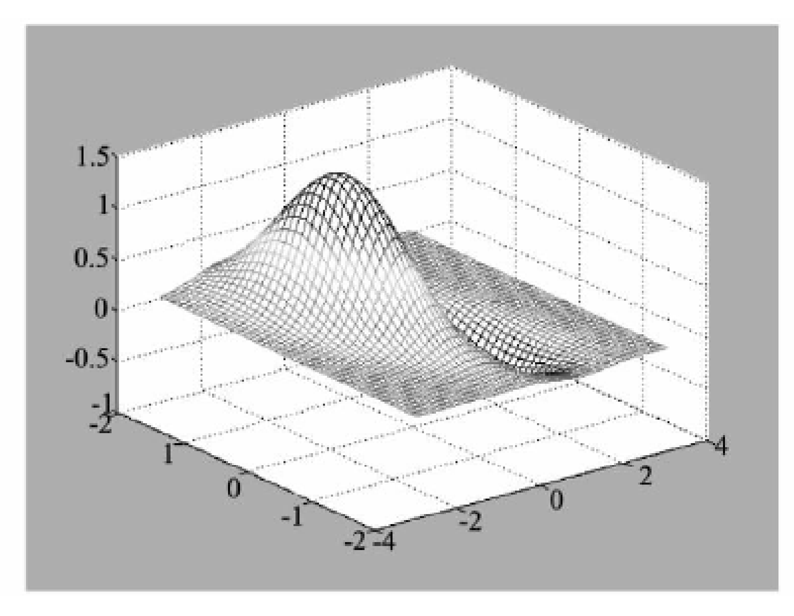


图 3.7　mesh 函数和 meshgrid 函数联合使用示例

3.1.3　三维曲面图

三维曲面图除了各线条之间的空档用颜色填充以外，其他和网格图看起来是一样的。这种图一般使用函数 surf 或者 surfc 来绘制。函数 surf 使用和函数 mesh 相同的调用语法。仍以 peaks 函数的绘制为例：

```
[x,y,z]=peaks(30);
surf(x,y,z);
grid;
xlabel(' x—axis '),ylabel(' y—axis '),zlabel(' z—axis ');
title(' SURF of PEAKS ')
```

这段程序在 Matlab 中执行后的显示结果如图 3.8 所示。

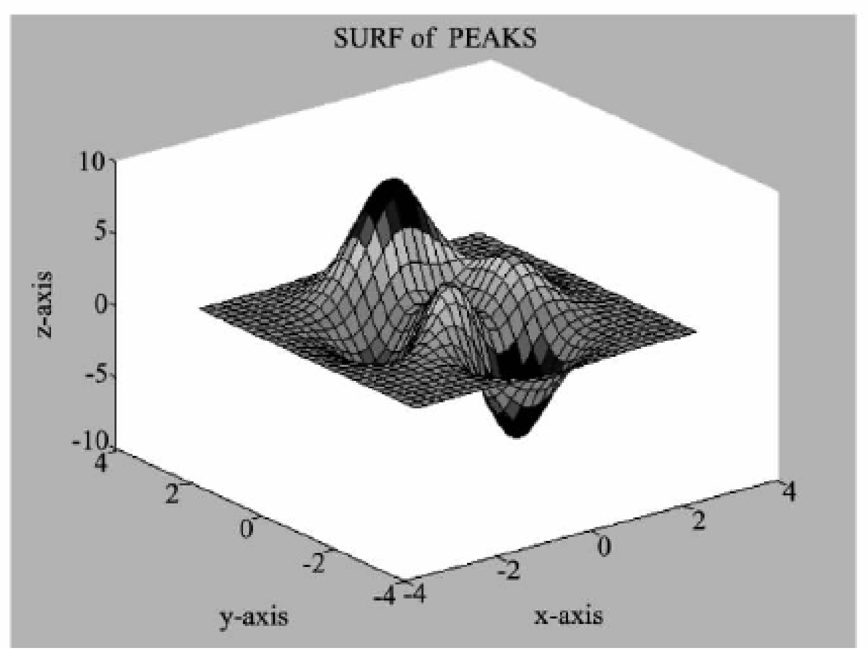


图 3.8　基本的三维曲面图形绘制

下例中参数 x 和 y 都是 41×41 的矩阵，由 meshgrid 函数产生，然后利用 subplot 函数划分坐标轴，比较 mesh 函数和 surf 函数的显示。

```
x=-2:0.1:2;
[x,y]=meshgrid(x,x);
r=sqrt(x.^2+x.^2)+eps;
z=sinc(r);
subplot(2,1,1);
mesh(z);
subplot(2,1,2);
surf(x,y,z);
```

这段程序在 Matlab 中执行后的显示结果如图 3.9 所示。

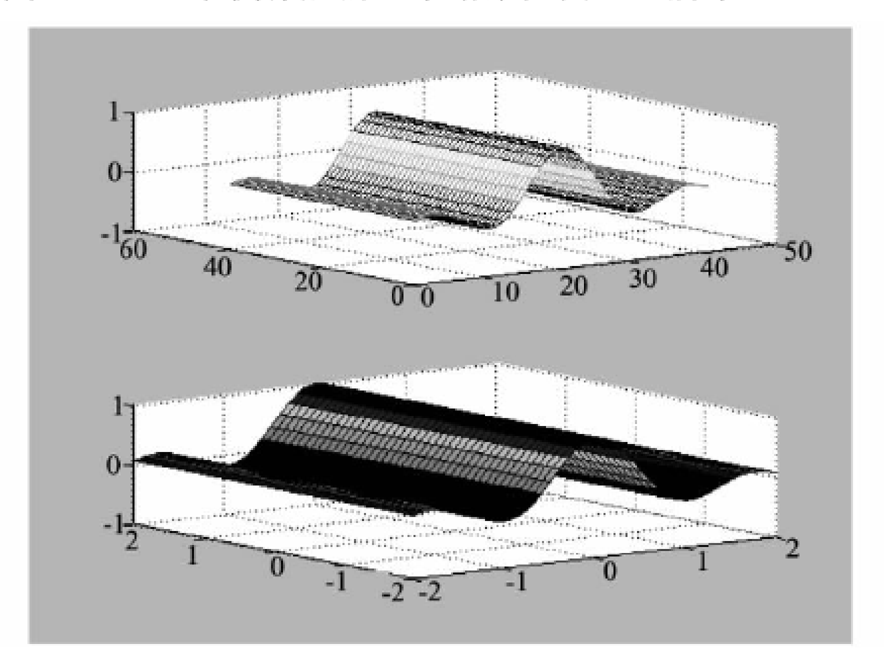


图 3.9 mesh 和 surf 的显示比较

surf 函数和 mesh 函数绘制出来的三维图实际上是一个 Matlab 图形对象，它有各种各样的属性，例如，其 MeshStyle 属性表示其网格的类型，既可以设置成水平的，又可以设置成垂直的。下面的代码将得出两个网格效果：

```
[x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
subplot(2,1,1);
h=surf(x,y,z);
axis([-3 3 -2 2 -0.7 1.5]);
set(h,'MeshStyle','row');
subplot(2,1,2);
h1=surf(x,y,z);
axis([-3 3 -2 2 -0.7 1.5]);
set(h1,'MeshStyle','column');
```

这段程序在 Matlab 中执行后的显示结果如图 3.10 所示。

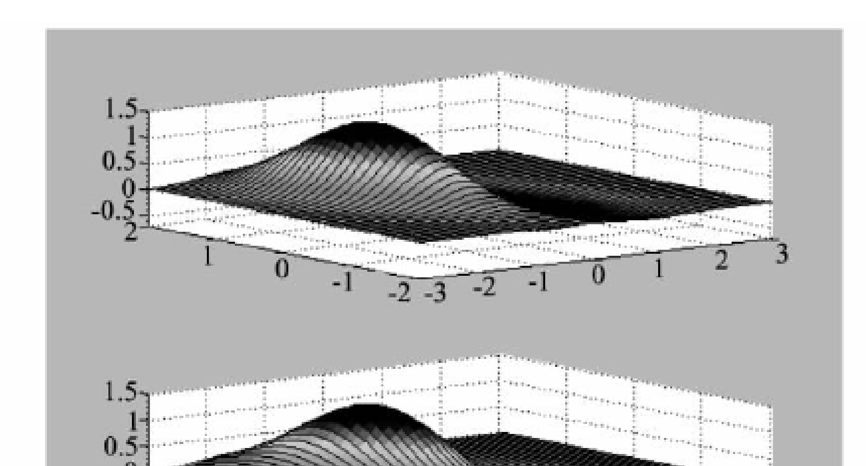


图 3.10 MeshStyle 属性示例

三维曲面图的一些特性和三维网格图相反：曲面图的线条是黑色的，线条之间的补片有颜色；在网格图里，补片是黑色的而线条有颜色。对函数 mesh，颜色沿着 z 轴按每一补片变化，而线条颜色不变。

在三维曲面图里，读者不必考虑像网格图一样隐蔽线条，但要考虑用不同的方法对表面加色彩。在前面的曲面图的例子中，就是分割成块，每块就像一块染色玻璃窗口或物体，黑线便是各单色染色玻璃块之间的连接。除此以外，Matlab 还提供了平滑加颜色和插值加颜色功能。这可以通过调用函数 shading 来实现。例如：

```
[x, y, z]=peaks(30);
surf(x, y, z)
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title(' SURF of PEAKS ')
shading faceted
```

这段程序在 Matlab 中执行后的显示结果如图 3.11 所示。

shading 函数有三个不同的参数选项，分别是 flat、faceted 和 interpolated-shaded，flat 将曲面划分为许多小块，每一块上的颜色是一致的；faceted 与 flat 的划分方式相同，但是它在块与块之间保留了黑线条作为分隔；interpolated-shaded 方式对表面块的着色也进行了插值处理，使其表面显得更光滑。下面是一个球面绘制中调用 shading 函数的例子：

```
subplot(2,2,1)
sphere(16)
axis square
```

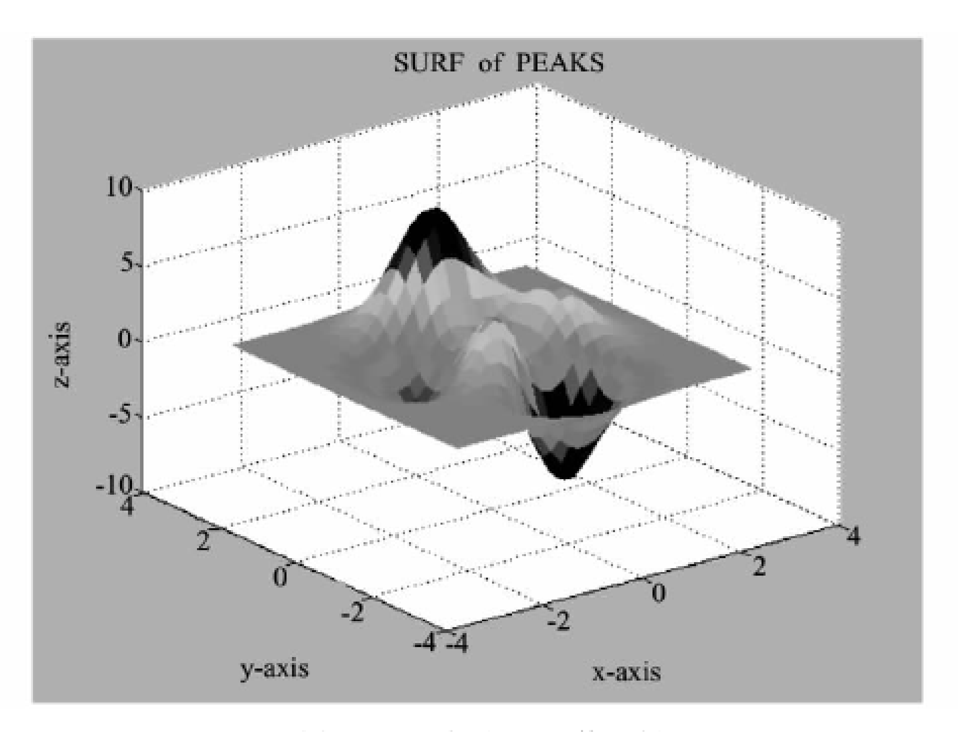


图 3.11　shading 函数示例

```
shading flat
title('Flat Shading')

subplot(2,2,2)
sphere(16)
axis square
shading faceted
title('Faceted Shading')

subplot(2,2,3)
sphere(16)
axis square
shading interp
title('Interpolated Shading')

subplot(2,2,4)
sphere(16)
axis square
title('Default Shading')
shading
```

这段程序在 Matlab 中执行后的显示结果如图 3.12 所示。

从最后一个球面可以看出，shading 函数默认的参数选项应该是 faceted，执行过程中 Matlab 会警告缺少参数，但仍然会执行这个 shading 函数。

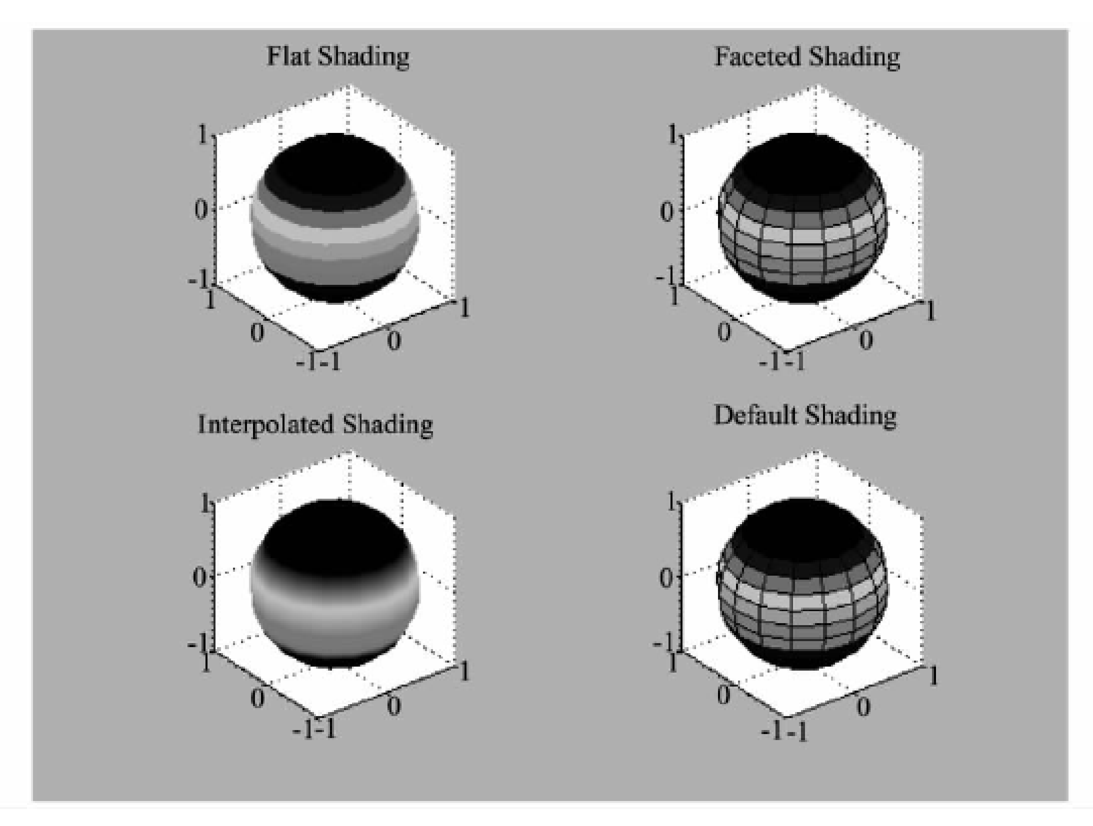


图 3.12　shading 函数选项示例

3.1.4　三维等值线图

Matlab 提供了另一种基本的三维图形，即三维等值线图，这种图形通过函数 contour3 来绘制。下面是一个最简单的例子。

```
[x,y,z]=peaks(30);
contour3(x,y,z,16)
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title(' CONTOUR3 of PEAKS ')
```

这段程序在 Matlab 中执行后的显示结果如图 3.13 所示。

contour3 函数的最后一个参数指定了等值线的数目。可以看出，图形中每一条线的颜色与二维函数 plot 的次序一致，这种颜色次序可以表现出明显的对比，但经常模糊了所代表的数据的一些重要特性。如果能使每一条线遵循在网格图和曲面图里所用的加色方法，那么效果会好得多，下面是改进的代码：

```
[X,Y,Z]=peaks(30);
N=16;
clf
view(3)
hold on
set(gca, 'ColorOrder' ,hsv(N))
```

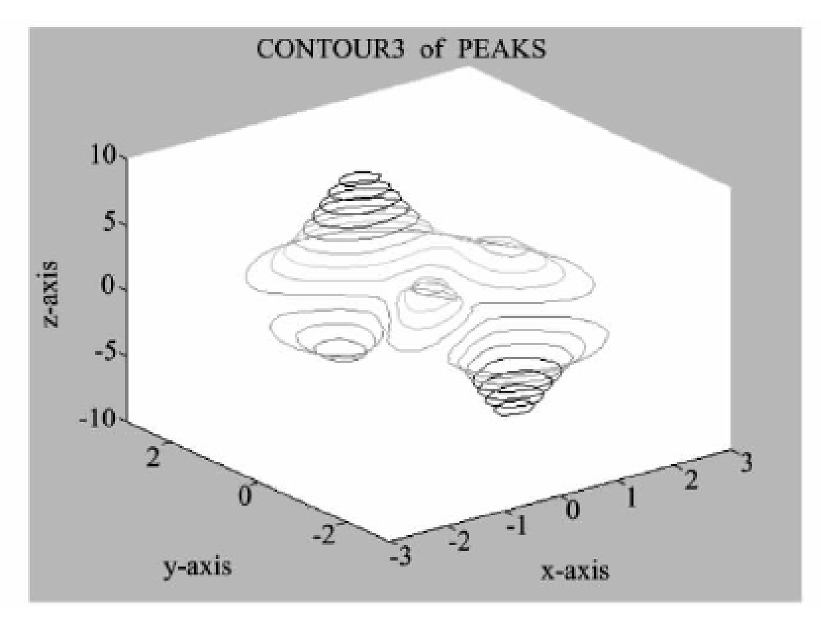


图 3.13　三维等值线图示例

```
contour3(X,Y,Z,N)
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title(' CONTOUR3 of PEAKS ')
hold off
```

这段程序在 Matlab 中执行后的显示结果如图 3.14 所示。

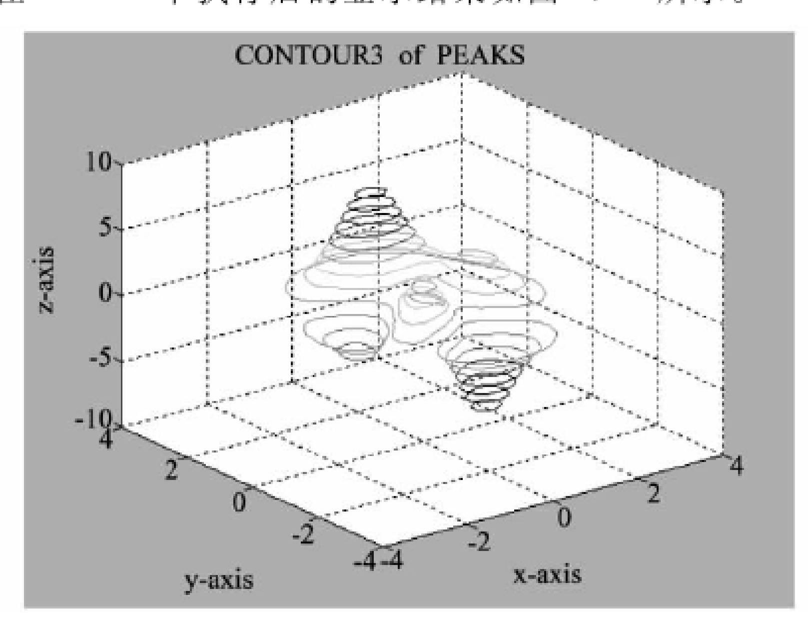


图 3.14　改进后的三维等值线图示例

在这个等值线图中,各条等值线的颜色沿着 z 轴的变化就和网格图及曲面图一样。

Matlab 提供的三维函数还包括 quiver、fill3 和 clabel,下面分别举例说明。函数 quiver 在等值线图上画出方向或速度箭头。例如:

```
[X,Y,Z]=peaks(16);
[DX,DY]=gradient(Z, .5, .5);
```

```
contour(X,Y,Z,10)
hold on
quiver(X,Y,DX,DY)
hold off
```

这段程序在 Matlab 中执行后的显示结果如图 3.15 所示。

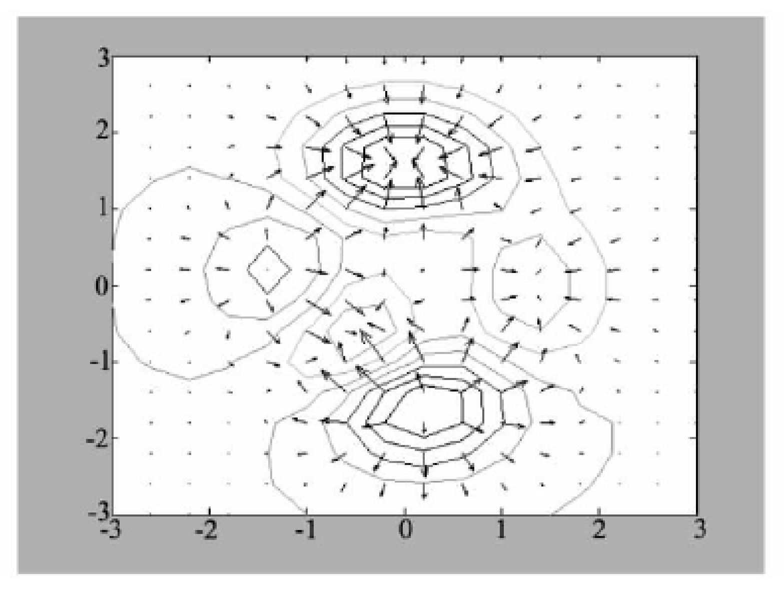


图 3.15 quiver 函数示例

函数 fill3 等效于二维函数 fill,可在三维空间内画出填充过的多边形。函数 fill3(x,y,z,c)使用数组 x、y 和 z 作为多边形的顶点而 c 指定了填充的颜色。例如,下例用随机的顶点坐标值画出五个黄色三角形:

```
fill3(rand(3,5),rand(3,5),rand(3,5),'y')
```

这段程序在 Matlab 中执行后的显示结果如图 3.16 所示。

函数 clabel 给等值线图标上高度值。使用时,函数 clabel 需要函数 contour 等值线矩阵的输出。

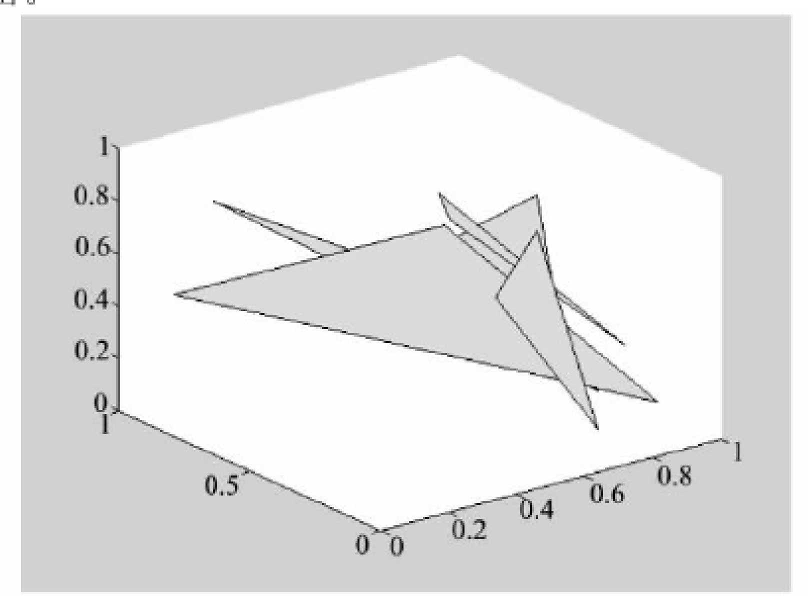


图 3.16 fill3 函数示例

```
[X,Y,Z]=peaks(30);
cs=contour(X,Y,Z,8); % request output from contour
clabel(cs) % add labels identifying heights
xlabel('X-axis'),ylabel('Y-axis')
title('CONTOUR of PEAKS with LABELS')
```

这段程序在 Matlab 中执行后的显示结果如图 3.17 所示。

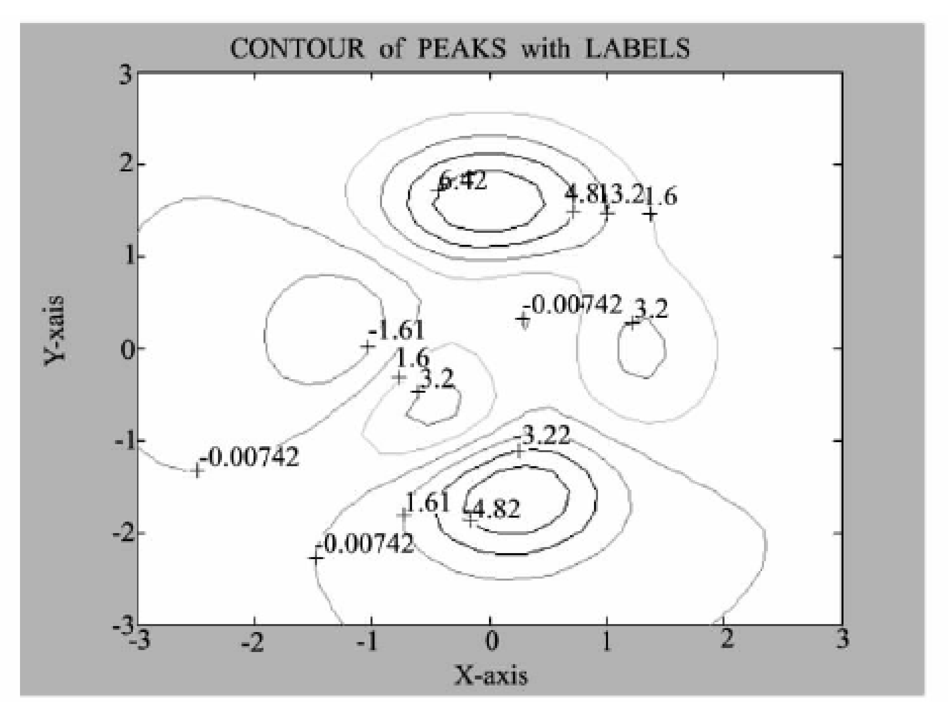


图 3.17 clabel 函数示例

3.2 三维图形的控制

3.2.1 控制图形视角

在 Matlab 中，三维图形的图形视角和二维图形一样都是由 view 函数来控制。例如，下面的语句使用 surf 函数绘制一个三维曲面：

```
[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2 -Y.^2);
grid;
xlabel('x-axis'),ylabel('y-axis'),zlabel('z-axis');
surf(X,Y,Z)
```

这段程序在 Matlab 中执行后的显示结果如图 3.18 所示。

使用下面的语句可以使视角沿着 y 轴的方向，并且位于 $z=0$ 处：

```
view([180 0])
```

这段程序在 Matlab 中执行后的显示结果如图 3.19 所示。

这里函数 view(az,el)将视角改变到所指定的方位角 az 和仰角 el。Matlab 中

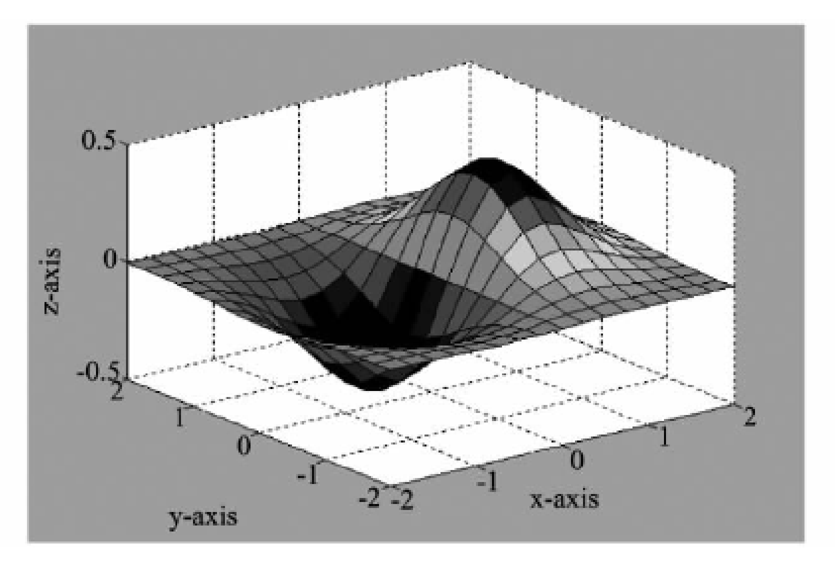


图 3.18　使用 view 函数控制视角

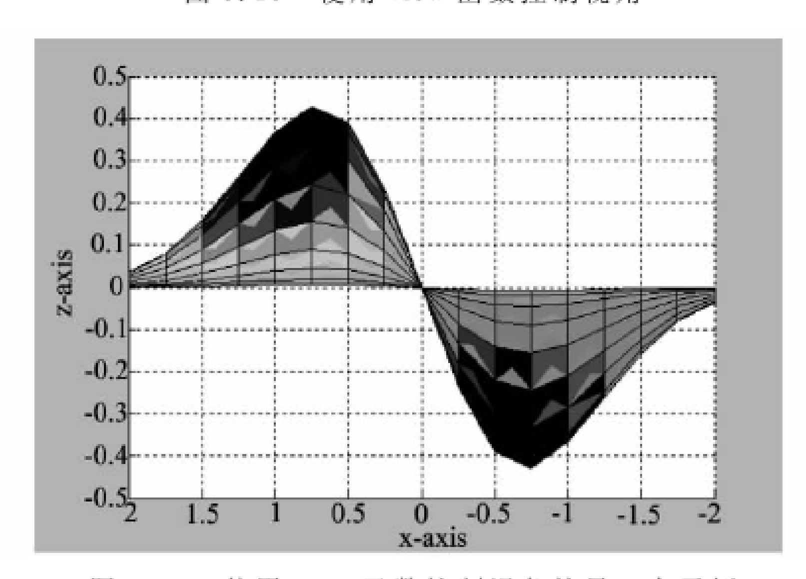


图 3.19　使用 view 函数控制视角的另一个示例

与 $z=0$ 平面所成的方向角称为仰角，而与 $x=0$ 平面的夹角称为方位角。默认的三维视角仰角为 30°，方位角为 −37.5°；而默认的二维视角仰角为 90°，方位角为 0 度。上面的例子中将方位角置为 180°，而仰角置为 0°，再观察下例中 view 函数对三维曲面的效果。

```
x=linspace(0,3*pi);
z1=sin(x);
z2=sin(2*x);
z3=sin(3*x);
y1=zeros(size(x));
y3=zeros(size(x));
y2=y3/2;

subplot(2,2,1)
plot3(x, y1, z1, x, y2, z2, x, y3, z3)
```

```
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title('Default Az = -37.5,E1 = 30 ')
view(-37.5,30)

subplot(2,2,2)
plot3(x, y1, z1, x, y2, z2, x, y3, z3)
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title( ' Az Rotated to 52.5 ')
view(-37.5+90,30)

subplot(2,2,3)
plot3(x, y1, z1, x, y2, z2, x, y3, z3)
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title( ' E1 Increased to 60 ' )
view(-37.5,60)

subplot(2,2,4)
plot3(x, y1, z1, x, y2, z2, x, y3, z3)
grid;
xlabel(' x-axis '),ylabel(' y-axis '),zlabel(' z-axis ');
title( ' Az = 0,E1 = 90 ' )
view(0,90)
```

这段程序在 Matlab 中执行后的显示结果如图 3.20 所示。

view 函数的用法见表 3.1。

表 3.1　view 函数的用法

view(az,el) view([az,el])	将视图设定为方位角 az 和仰角 el
view([x,y,z])	在笛卡儿坐标系中将视图设为沿向量[x,y,z]指向原点,如 view([0,0,1])=view(0,90)
view(2)	设置默认的二维视角,az=0,el=90
view(3)	设置默认的三维视角,az=-37.5,el=30
[az,el]=view	返回当前的方位角 az 和仰角 el
view(T)	用一个 4×4 的转矩阵 $\boldsymbol{T}$ 来设置视角
T=view	返回当前的 4×4 转矩阵

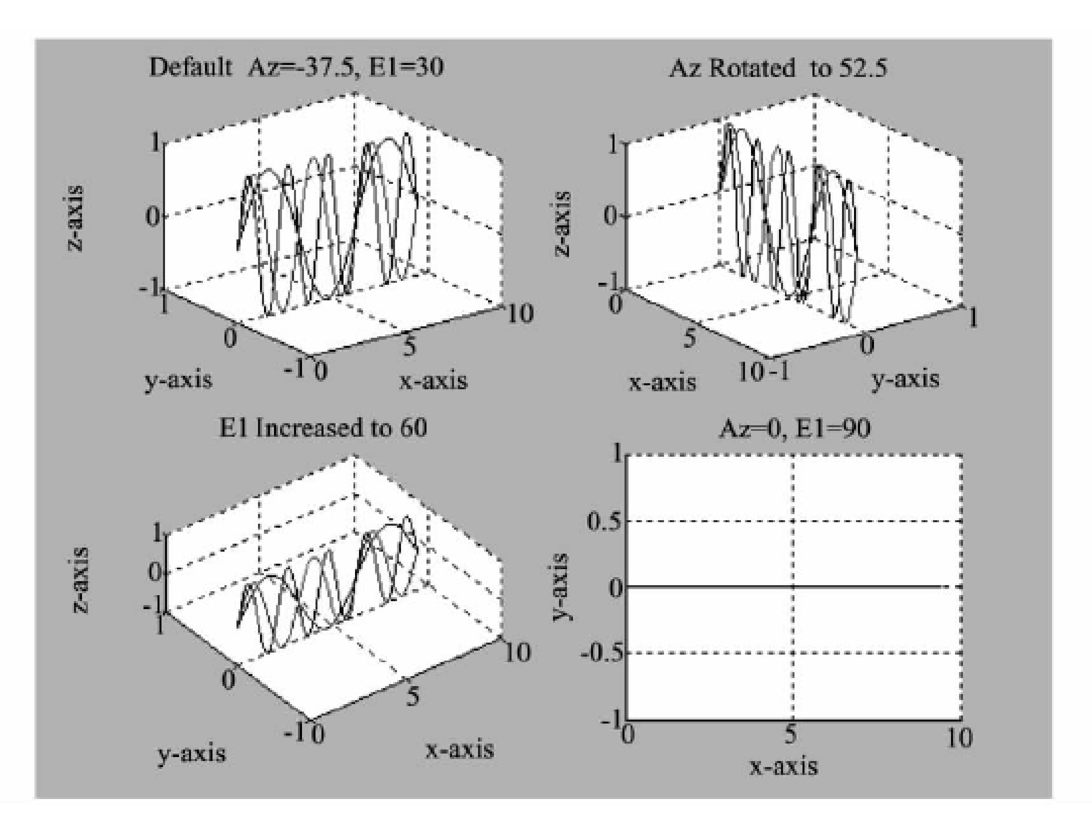


图 3.20 view 函数对三维曲面的效果

3.2.2 控制光照

Matlab 使用 light 函数来控制光照，light 函数的用法如下：

```
light('PropertyName',PropertyValue,...)
handle=light(...)
```

light 函数为当前的坐标轴对象创建一个光照对象，它仅仅对坐标轴中的 patch 和 surface 对象产生影响。light 函数有三个重要的属性：

(1) Color——光照的颜色。

(2) Style——光源的模式，默认值是无限远。

(3) Position——对于无限远的情况下，这个值表示光源的方向，对于有限距离的光源，这个值给出了光源的坐标。

下面的例子中，为三维曲面添加的光照对象被指定为有限距离的光源，并给出了光源的三维坐标。

```
membrane
light('Position',[0 -2 1])
```

这段程序在 Matlab 中执行后的显示结果如图 3.21 所示。

light 函数可以增强某些函数曲面的表面效果，如下例中使用 ezsurf 来模拟 $-6\pi\sim6\pi$ 上的函数 $\sin\sqrt{x^2+y^2}/\sqrt{x^2+y^2}$，可以使用如下命令：

```
ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)',[-6*pi,6*pi])
```

这句程序在 Matlab 中执行后的显示结果如图 3.22 所示。

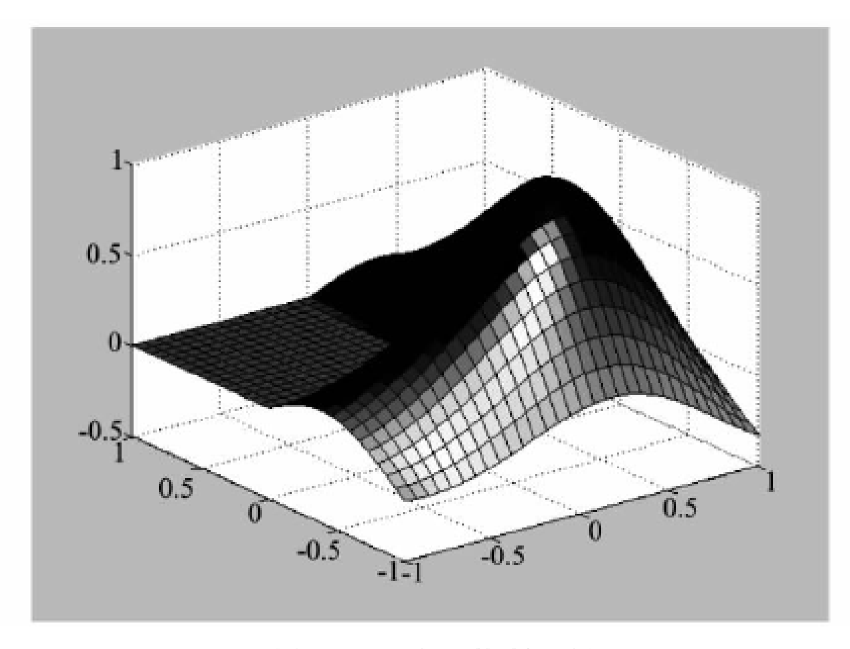


图 3.21　光照控制示例

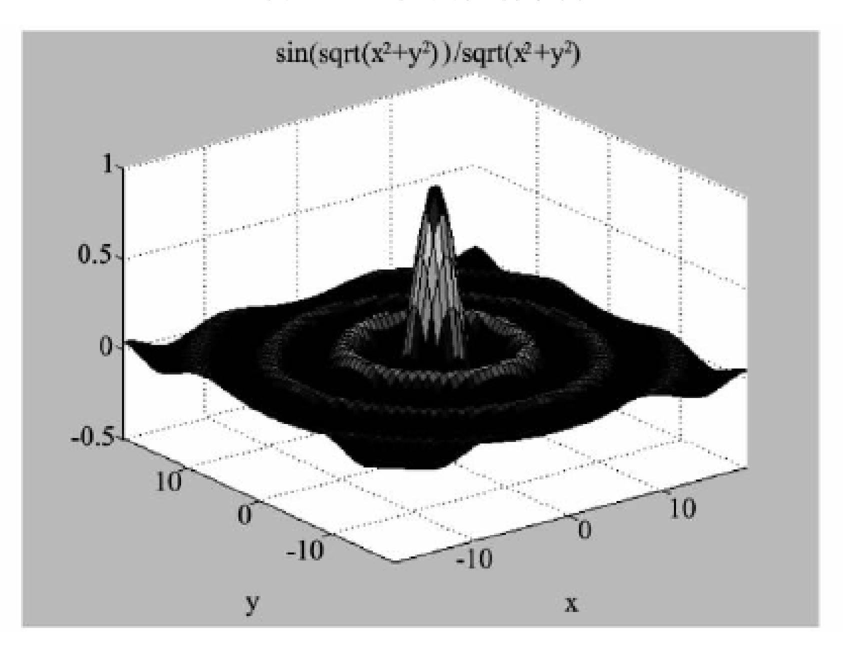


图 3.22　光照控制的另一个示例

接着使用 lightangle 函数添加光照，指定方位角为 0°，仰角为 75°，并使用 shading 函数增强表面。

```
view(0,75)
shading interp
lightangle(-45,30)
set(gcf,'Renderer','zbuffer')
set(findobj(gca,'type','surface'),
  'FaceLighting','phong',
  'AmbientStrength',.3,'DiffuseStrength',.8,
  'SpecularStrength',.9,'SpecularExponent',25,
  'BackFaceLighting','unlit')
```

这段程序在 Matlab 中执行后的显示结果如图 3.23 所示。

图 3.23　使用 lightangle 函数添加光照

上面的代码使用 findobj 函数获得了 surface 对象的句柄，然后就可以使用这个句柄设置图形的属性，影响光照的 surface 对象的属性有 13 个，有关这些属性的详细情况可参看 Matlab 的随机文档。

Matlab 中的三维绘图函数和三维绘图工具见表 3.2 和表 3.3。

表 3.2　三维绘图函数列表

三维绘图函数	说　明	三维绘图函数	说　明
contour	二维等值线图，即从上向下看的 contour3 等值线图	plot3	直线图
contour3	等值线图	quiver	二维带方向箭头的速度图
fill3	填充的多边形	surf	曲面图
mesh	网格图	surfc	具有基本等值线图的曲面图
meshc	具有基本等值线图的网格图	surfl	带亮度的曲面图
meshz	有零平面的网格图	waterfall	无交叉线的网格图
pcolor	二维伪彩色绘图，即从上向下看的 surf 图		

表 3.3　三维绘图工具列表

三维绘图工具	说　明	三维绘图工具	说　明
axis	修正坐标轴属性	figure	创建或选择图形窗口
clf	清除图形窗口	getframe	捕捉动画帧
clabel	放置等值线标签	grid	放置网格
close	关闭图形窗口	griddata	对画图用的数据进行内插

（续）

三维绘图工具	说 明	三维绘图工具	说 明
hidden	隐蔽网格图线条	text	在指定的位置放文本
hold	保留当前图形	title	放置标题
meshgrid	产生三维绘图数据	view	改变图形的视角
movie	放动画	xlabel	放置 x 轴标记
moviein	创建帧矩阵，存储动画	ylabel	放置 y 轴标记
shading	在曲面图和伪彩色图中用分块、平滑和插值加阴影	zlabel	放置 z 轴标记
subplot	在图形窗口内画子图		

3.3 动　画

Matlab 除了支持二维和三维基本图形的显示之外，还可以将一系列基本图形存储起来，然后把它们按次序重放出来。在某种意义上，动画提供的运动为图形增加了另一个维数。通常图形的次序不必以任意的方式关联起来。一种明显的动画类型是取出三维图形然后缓慢地将它旋转，这样就可以从不同角度来观察它。另一种类型是当一个参数变化时，依次显示某些问题解的图形。

Matlab 中的函数 moviein、getframe 和 movie 提供了捕捉和播放动画所需的工具。函数 moviein 可以产生一个帧矩阵来存放动画中的帧；函数 getframe 对当前的图形进行快照；而函数 movie 按顺序回放各帧。照这样，捕捉和回放动画的方法是：

（1）创建帧矩阵；

（2）对动画中的每一帧生成图形，并把它捕捉到到帧矩阵里；

（3）从帧矩阵里回放动画。

例如，下面的一段 M 文件脚本，其中绘制了函数 peaks 并且将它绕 z 轴旋转。

```
% movie making example: rotate a 3-D surface plot

    [X,Y,Z]=peaks(30); % create data
    surfl(X,Y,Z) % plot surface with lighting
    axis([-3 3 -3 3 -10 10]) % fix axes so that scaling does not change
    axis off % erase axes because they jump around
    shading interp % make it pretty with interpolated shading
    colormap(hot) % choose a good colormap for lighting
```

```
m=moviein(15); % choose 15 movie frames for frame matrix m

for I=1:15 % rotate and capture each frame
    view(-37.5+24*(i-1),30) % change viewpoint for this frame
    m(:,i)=getframe; % add figure to frame matrix
end

movie(m) % play the movie!
```

注意，动画中的每一帧在帧矩阵中占据一个不同的列。帧矩阵的大小随着动画中的帧数和图形窗口的大小而改变，但与所绘图形的复杂性无关，这是因为函数 getframe 仅仅是捕捉位图。缺省情况下，函数 movie 只放一遍动画。通过加入其他输入参量，它可以向前放、向后倒放、放指定次数或按特定的帧速率播放。

另外一种创建动画的办法是不断在屏幕上对图形对象进行 erase 和 redraw，而每次 redraw 时对对象的某个参数进行增量变化，通常这种方法需要用到图形对象的 EraseMode 属性，下面以一个例子来说明这个属性的用法。

```
n=20
s = .02
x = rand(n,1)-0.5;
y = rand(n,1)-0.5;
h = plot(x,y,'.');
axis([-1 1 -1 1])
axis square
grid off
set(h,'EraseMode','xor','MarkerSize',18)
```

首先产生 20 对随机坐标，并将这些点的横纵坐标限制在 -1 到 1 之间，然后将它们显示在图形窗口中，再使用 set 指令将 plot 句柄的 EraseMode 设为“xor”，这样做的目的在于，如果点集中某个点的坐标发生变化，则不必重画点集中所有的点，而只需要将变化的点重新显示在图中即可。

下面的语句是一个无限循环，每个循环体对坐标进行微调，这样显示的效果就如同动画一样。

```
while 1
  drawnow
  x = x + s*randn(n,1);
  y = y + s*randn(n,1);
  set(h,'XData',x,'YD ata',y)
end
```

结束循环使用 Ctrl－C 即可，显示结果如图 3.24 所示。

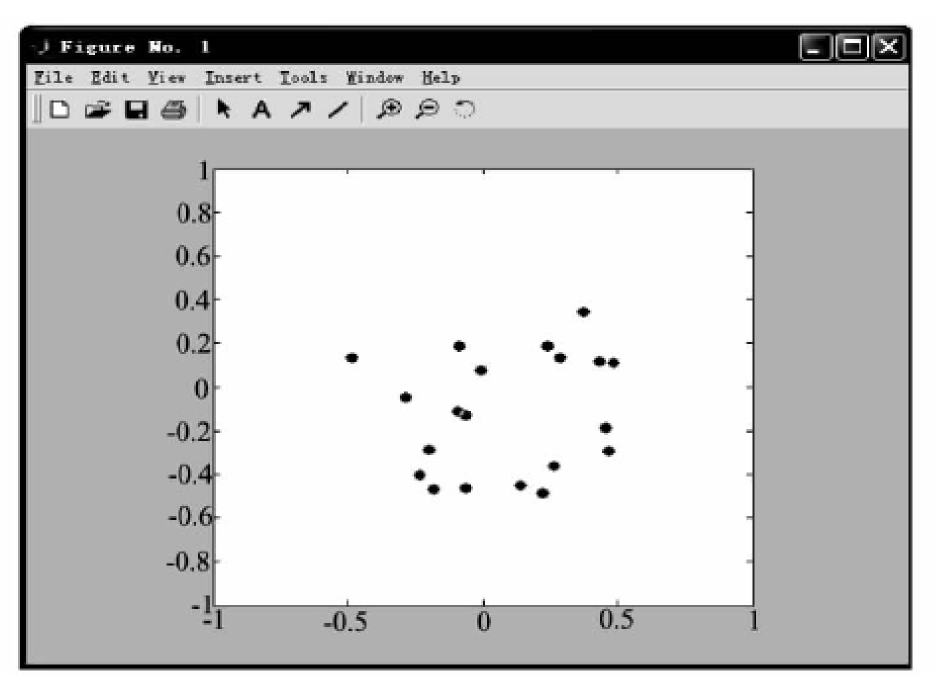


图 3.24　简单的动画示例

第 4 章　图像的运算

4.1　图像的代数运算

图像的代数运算是基本代数运算在图像上的实现，最简单的方法是直接使用 Matlab 的代数运算符来实现图像代数运算，但这样做之前必须将图像转换为与基本代数运算符类型相容的双精度浮点类型，而 Matlab 的图像处理工具包为用户提供了适合所有非稀疏数值类型的代数运算的函数集合。这些函数见表 4.1。

表 4.1　图像的代数运算函数

函数名	说　明	函数名	说　明
imabsdiff	两张图像的绝对值差	imlincomb	两张图像的线性组合
imadd	两张图像的代数加法	immultiply	两张图像的代数乘法
imcomplement	一张图像的补运算	imsubtract	两张图像的代数减法
imdivide	两张图像的代数除法		

这些函数在本节将一一结合实例说明。

4.1.1　绝对值差函数 imabsdiff

函数 imabsdiff 用于计算两张图像的绝对值差，其使用格式如下：

```
Z = imabsdiff(X,Y)
```

imabsdiff 函数将相同类型、相同长度的数组 X 和 Y 的对应位分别做减法，返回的结果是每一位差的绝对值，即返回的数组 Z 应该和 X、Y 的类型相同，如果 X、Y 是整数数组，那么结果中超过整数类型范围的部分将被截去；如果 X、Y 是浮点数组，用户也可以使用基本运算 abs($X-Y$)来代替这个函数。

下例计算两个 uint8 类型的数组的绝对值差：

```
X=uint8([ 255 10 75; 44 225 100]);
Y=uint8([ 50 50 50; 50 50 50 ]);
Z=imabsdiff(X,Y)
```

```
Z=
   205     40     25
   6      175     50
```

下例显示了一张经过处理后的图和原图的差别：

```
I=imread('cameraman.tif');
J=uint8(filter2(fspecial('gaussian'), I));
K=imabsdiff(I,J);
imshow(K,[])     %[]=scale data automatically
```

这段程序在 Matlab 中执行后的显示结果如图 4.1 所示。




图 4.1 绝对值差函数 imabsdiff 示例

4.1.2 图像的叠加函数 imadd

imadd 函数用于产生两张图像的叠加效果，其使用格式如下：

```
Z = imadd(X,Y)
```

其中 X、Y 是类型相同的数组。下例是一个简单的例子，说明了对 uint8 数组进行相加得到的结果：

```
X=uint8([ 255 0 75; 44 225 100]);
Y=uint8([ 50 50 50; 50 50 50 ]);
Z=imadd(X,Y)
Z=

   255     50     125
   94     255     150
```

下例对两张图像进行了叠加：

```
I=imread('rice.tif');
J=imread('cameraman.tif');
```

```
K=imadd(I,J,'uint16');
imshow(K,[])
```

这段程序在 Matlab 中执行后的显示结果如图 4.2 所示。



图 4.2 叠加函数 imadd 示例

实际上 imadd 函数可以通过指定常数参数增强一张图像的亮度。下面还是以 rice. tif 这张图像作为例子：

```
I=imread('rice.tif');
J=imadd(I,50);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

这段程序在 Matlab 中执行后的显示结果如图 4.3 所示。

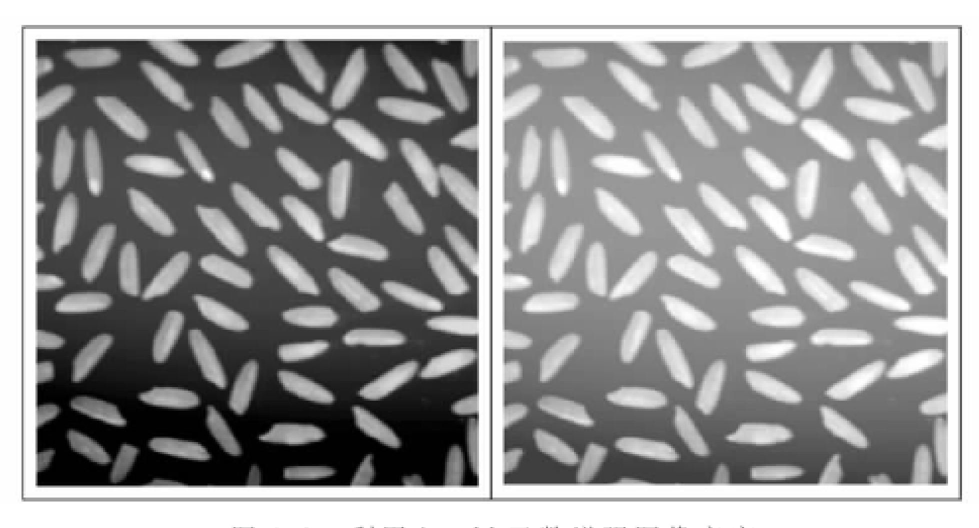


图 4.3 利用 imadd 函数增强图像亮度

4.1.3 图像求补函数 imcomplement

imcomplement 函数适用于各种图像格式，如果是二进制图像，那么函数将对图像的每一位求补；如果是 RGB 格式的图像，函数将会用像素的最大值减去图像的原始值，得到输出图像相应位置的值。其使用格式如下：

```
IM2 = imcomplement(IM)
```

IM 和 IM2 分别是原始图像和输出图像，其类型可以是 Matlab 支持的各种图像类型。

下例说明对一个 uint8 类型的数组求补的结果：

```
X = uint8([ 255 10 75; 44 225 100]);
X2=imcomplement(X)
X2=
    0     245   180
    211   30    155
```

下例将一张二进制的图像求补：

```
bw=imread('text.tif');
bw2=imcomplement(bw);
subplot(1,2,1),imshow(bw)
subplot(1,2,2),imshow(bw2)
```

这段程序在 Matlab 中执行后的显示结果如图 4.4 所示。

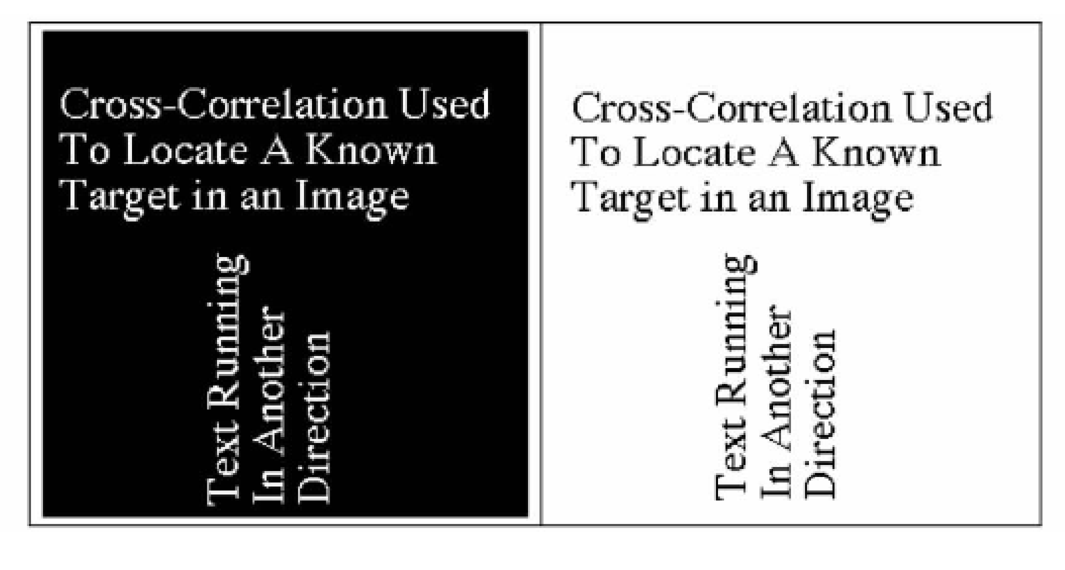


图 4.4　二进制图像的求补

第三个例子是对灰度图像进行补运算：

```
I=imread('bonemarr.tif');
J=imcomplement(I);
imshow(I), figure, imshow(J)
```

这段程序在 Matlab 中执行后的显示结果如图 4.5 所示。

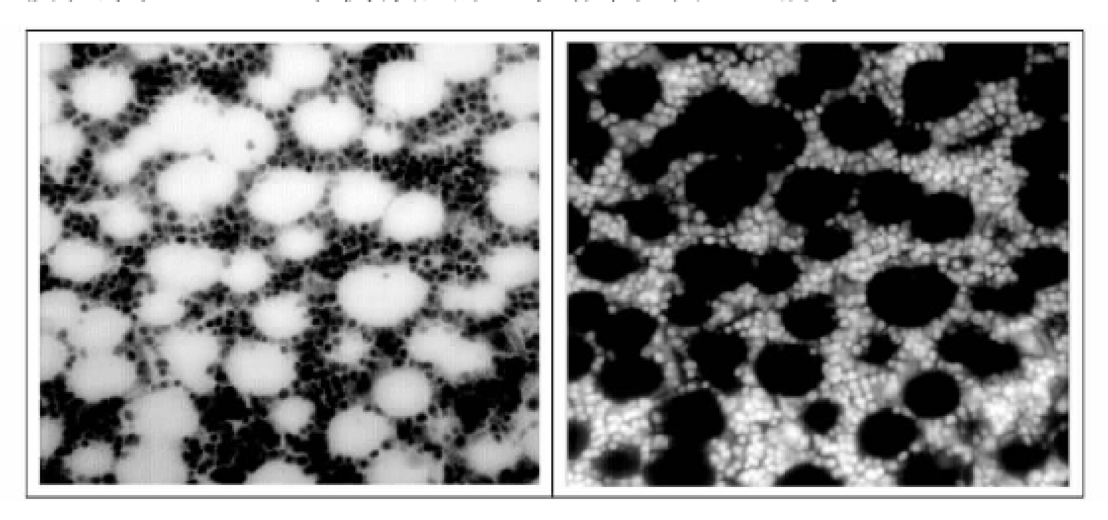


图 4.5　灰度图像的补运算

4.1.4　图像的除法运算 imdivide

imdivide 函数的运算法则与前面介绍的函数类似，不同的是这个函数还可以

以常数作为参数，即将一张图像与一个常数做除法运算，其使用格式如下：

```
Z = imdivide(X,Y)
```

图像的除法运算通常用于校正成像设备的非线性误差，也可以用于检测两张图像之间的差别，它给出的是相应像素值的变化比率，而不是每个像素的绝对差异。

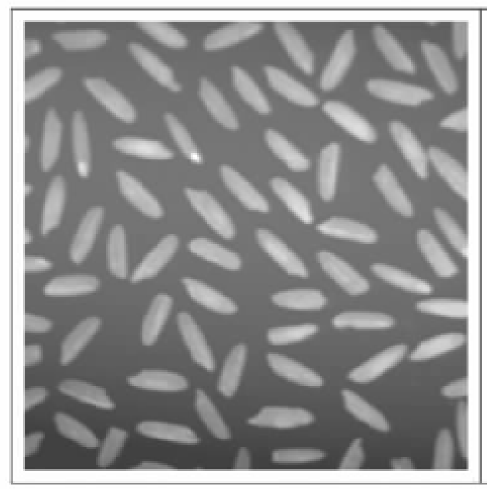
下例说明对 uint8 类型的数组进行除法操作得到的结果：

```
X=uint8([ 255 10 75; 44 225 100]);
Y=uint8([ 50 20 50; 50 50 50 ]);
Z=imdivide(X,Y)
Z=
     5   1   2
     1   5   2
```

下例说明了除法运算对图像的变换效果：

```
I=imread('rice.tif');
blocks=blkproc(I,[32 32],'min(x(:))');
background=imresize(blocks,[256 256],'bilinear');
Ip=imdivide(I,background);
imshow(Ip,[]) %[]=let imshow scale data automatically
```

这段程序在 Matlab 中执行后的显示结果如图 4.6 所示。



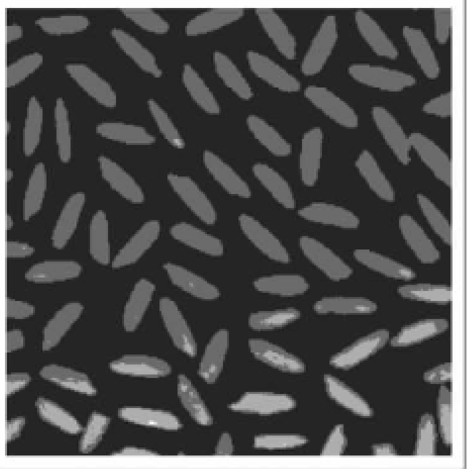


图 4.6　除法运算对图像的变换效果

如果将参数改为常数，那么结果如下：

```
I=imread('rice.tif');
J=imdivide(I,2);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

这段程序在 Matlab 中执行后的显示结果如图 4.7 所示。

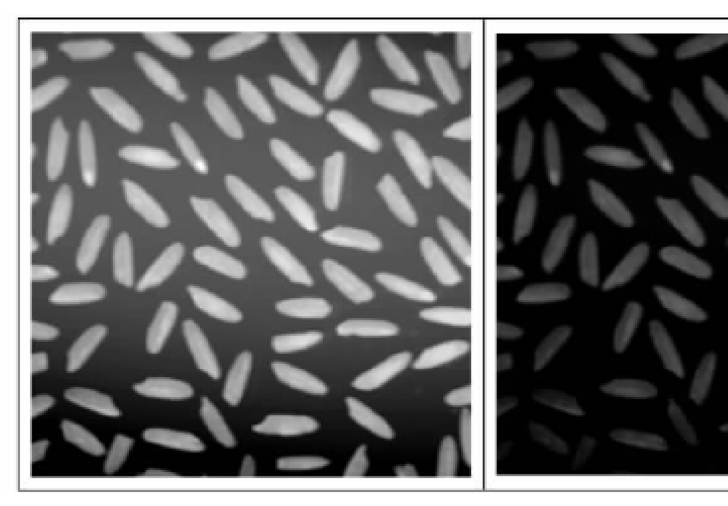


图 4.7 参数为常数的除法运算

4.1.5 线性组合函数 imlincomb

imlincomb 函数的功能是对两张图像进行线性组合操作，其使用格式如下：

```
Z=imlincomb(K1,A1,K2,A2,...,Kn,An)
Z=imlincomb(K1,A1,K2,A2,...,Kn,An,K)
Z=imlincomb(..., output_class)
```

其中所有的 Ki 为实数类型双精度浮点标量值，Ai 为同类型的数组，而 K 为实数类型双精度的常数，这个函数还可以以字符串的形式指定输出的类型。

下例中使用了 imlincomb 函数对图形进行符合下列公式的线性组合操作：

$$K(r,c)=I(r,c)-J(r,c)+128$$

其代码如下：

```
I=imread('cameraman.tif');
J=uint8(filter2(fspecial('gaussian'), I));
K=imlincomb(1,I,-1,J,128);
imshow(K)
```

这段程序在 Matlab 中执行后的显示结果如图 4.8 所示。

图 4.8 线性组合函数 imlincomb 示例

线性运算函数通常可以用来代替多个简单代数函数使用，以完成相同图像的四则运算操作。

4.1.6 图像的乘法操作 immultiply

immultiply 函数用于对两张图像进行掩模操作，即屏蔽掉图像的某些部分。使用乘法函数时通常需要指定缩放参数，如果大于 1 则增强图像的亮度，反之则减弱图像的亮度。其调用格式如下：

```
Z = immultiply(X,Y)
```

immultiply 函数对两张图像相应的像素值进行点乘，并将乘法的运算结果作为输出图像相应的像素值。下例对图像进行亮度的增强：

```
I=imread('moon.tif');
J=immultiply(I,0.5);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

这段程序在 Matlab 中执行后的显示结果如图 4.9 所示。



图 4.9　图像的掩模操作示例

在使用乘法函数的时候，对 uint8 图像进行操作往往会发生溢出现象，immultiply 函数将溢出的数据截取为数据类型允许的最大值。为了避免这种现象，一般在乘法运算前将 unit8 图像转换为一种数据范围更大的图像类型。

4.1.7 图像的减法函数 imsubtract

imsubtract 函数对两张图像进行代数减法，通常用于检测图像变化以及运动

物体的图像处理。其使用格式与加法函数类似，下例是两个 uint8 类型的数组相减的例子：

```
X=uint8([ 255 10 75; 44 225 100]);
Y=uint8([ 50 50 50; 50 50 50 ]);
Z=imadd(X,Y)
Z=


  205   0   25
  0   175   50
```

下例先根据原始图像生成其背景亮度图像，然后再从原始图像中将背景亮度图像减去。

```
I=imread('rice.tif');
blocks=blkproc(I,[32 32],'min(x(:))');
background=imresize(blocks,[256 256],'bilinear');
Ip=imsubtract(I,background);
imshow(Ip,[])
```

这段程序在 Matlab 中执行后的显示结果如图 4.10 所示。

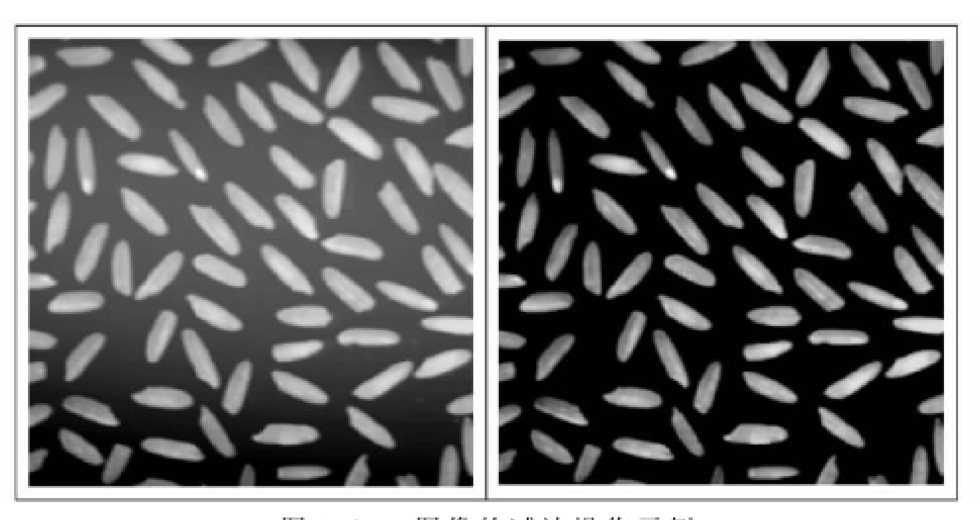


图 4.10　图像的减法操作示例

4.2 几何操作

Matlab 提供的图像几何操作函数是数字图像处理工具包的基本工具之一，这些函数包括图像尺寸大小变化、旋转、错切、删除或增加图像线条以及反射等，本节中将分别予以介绍。

4.2.1 改变图像大小

Matlab 使用 imresize 函数来改变图像的大小，其使用格式如下：

Y＝imresize(X ，M ，Method)

其中X表示需要进行操作的图像，M是放大的倍数，一般来说M是大于0的实数，如果M大于1则表示需要对图像进行放大，小于1则表示缩小图像。用户可以指定Method参数来选择使用何种插值方法。表4.2列出了method参数各个值及其含义。

表4.2　method函数参数列表

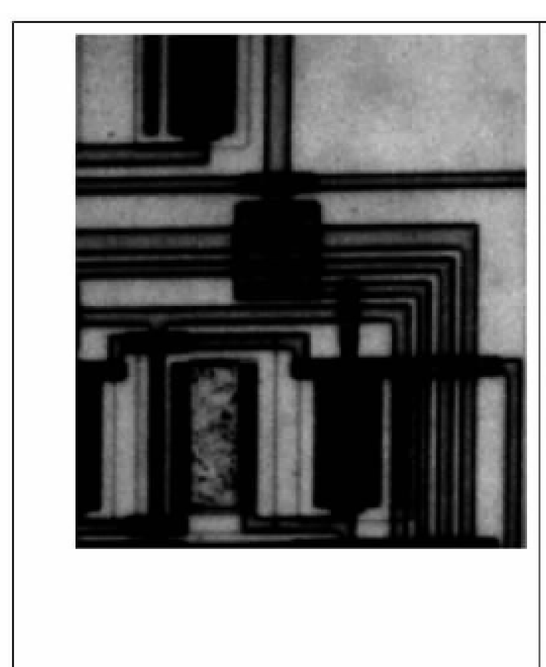
参数值	插值方法
'nearest'	最近邻插值法
'bilinear'	双线性插值法
'bicubic'	双三次插值法

imresize函数在默认情况下使用的是最近邻插值法。在这种算法中每一个插值输出像素的值是在输入图像中与其最邻近的采样点的值，这种插值方法的运算量最小，适用于索引色图像。而双线性插值的输出像素值是它在输入图像中2×2邻域采样点的平均值，它根据像素周围4个像素的灰度值在水平和垂直两个方向上对其进行插值。双三次插值使用三次插值函数，选择4×4的邻域采样点，取得的效果较好，但相应的计算量也比前两种要大。

下例指定了放大倍数为1.25，并使用默认的最近邻插值方法。

```
I=imread('circuit.tif')；
J=imresize(I,1.25)；
imshow(I)
figure, imshow(J)
```

这段程序在Matlab中执行后的显示结果如图4.11所示。



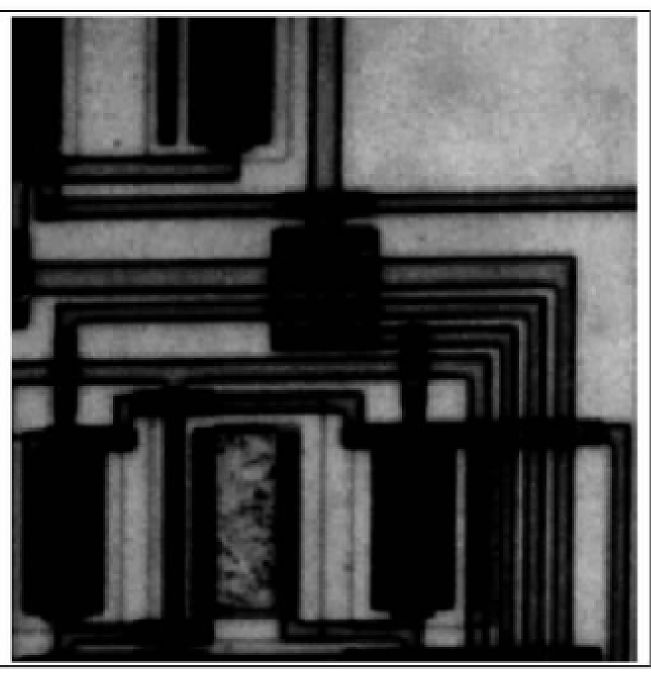


图4.11　imresize函数示例

用户还可以指定图像输出的大小，这需要传递一个向量参数给imresize函数，其中参数的两个分量分别指定了输出图像的行数和列数。如下例中，通过调用imresize函数将原图像X输出为200×480的图像。

```
Y=imresize(X,[200 480])
```

如果选择双线性插值或双三次插值来缩小一张图像，imresize函数会在插值操作之前调用低通滤波器对图像进行滤波，这项操作可以在一定程度上消除重采样过程中出现的波纹影响。

4.2.2 图像的旋转

Matlab提供了imrotate函数来对图像进行旋转操作，其调用方法如下：

```
Y=imrotate(X,angle,method)
Y=imrotate(X,angle,method,'crop')
```

其中X表示原图像，angle表示旋转的角度，这个角度是以逆时针的方向来计算的，负值则表示顺时针方向的旋转。method参数与前面介绍的imresize中的参数的意义相同。而crop参数表示将旋转以后的图像取与原图像大小相同的中心部分输出。

```
I=imread('circuit.tif');
J=imrotate(I,-15,'bilinear');
K=imrotate(I,-15,'bilinear','crop');
imshow(I)
figure, imshow(J)
figure, imshow(K)
```

这段程序在Matlab中执行后的显示结果如图4.12所示。

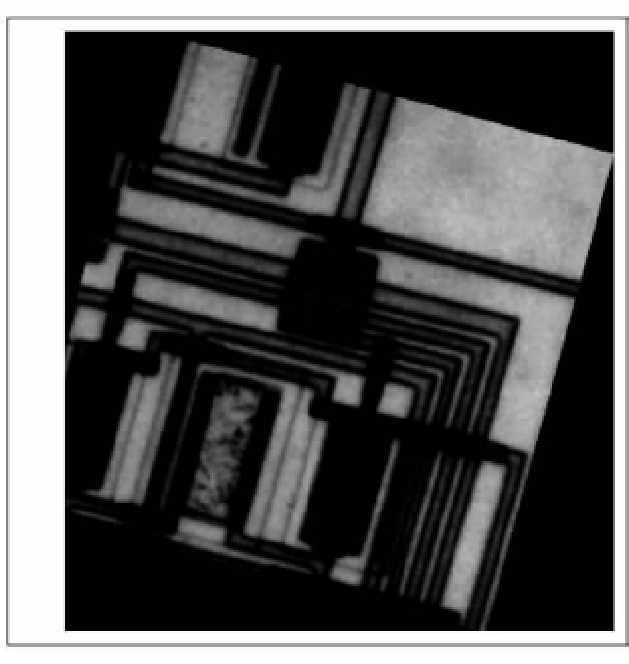



图4.12 imrotate函数示例

在未增加crop参数的情况下，图像在旋转操作后会在外部自动用黑色背景填满整个矩形区域，而指定了这个参数则返回一张裁剪过的与原始图像同样大小的旋转图像。

4.2.3 图像的裁剪

Matlab 提供 imcrop 函数对图像进行裁剪，函数的返回值是原图像的某个矩形区域，这个函数有两个基本参数：

(1) 待裁剪的原图像；

(2) 定义矩形裁剪区域的坐标点集。

下面是 imcrop 函数的使用格式：

```
I2=imcrop(I)
X2=imcrop(X,map)
RGB2=imcrop(RGB)

I2=imcrop(I,rect)
X2=imcrop(X,map,rect)
RGB2=imcrop(RGB,rect)

[...]= imcrop(x,y,...)
[A,rect]=imcrop(...)
[x,y,A,rect]=imcrop(...)
```

在上述这些用法中，如果不指定原图像时，imcrop 函数将当前坐标轴中的图像作为待剪切的图像，map 参数表示原图像为索引图像时的调色板。rect 参数则定义了图像剪切区域的矩形坐标，如果不指定坐标，则可以在原图像中使用拖放的方式来手动选择一个矩形区域，imcrop 函数将根据这个矩形区域产生一个新的图像，如图 4.13 所示。

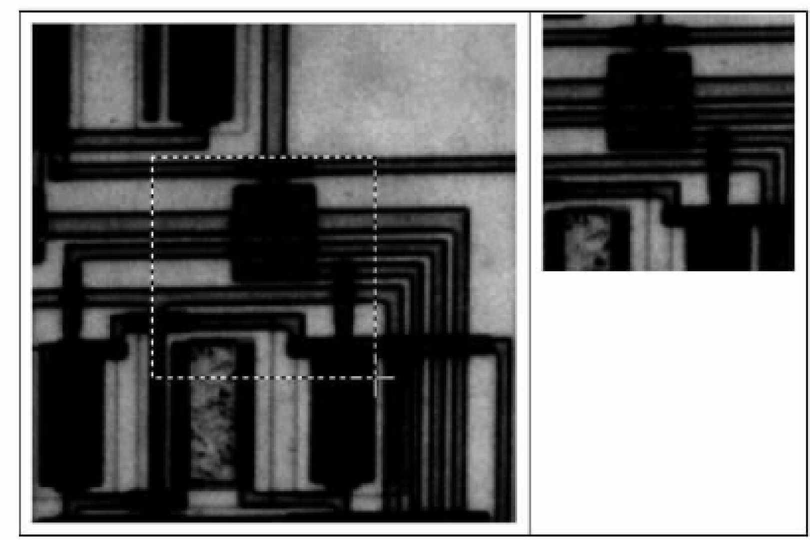


图 4.13 imcrop 函数示例

如果需要精确选择图像剪切的区域，可以直接指定矩形区域的坐标，例如：

```
I=imread('ic.tif');
```

```
I2=imcrop(I,[30 60 120 160]);
imshow(I);
figure,imshow(I2)
```

这段程序在 Matlab 中执行后的显示结果如图 4.14 所示。

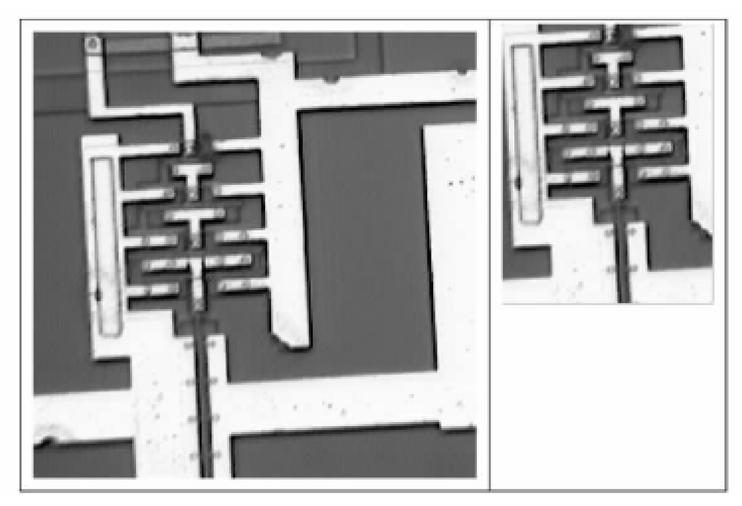


图 4.14　指定矩形区域的坐标

4.3　图像的邻域和块操作

4.3.1　滑动邻域操作

邻域操作是将每个输入的像素值及其某个邻域的像素值结合处理而得到对应的输出像素值的过程。通常这里指的邻域是一个远小于原图像尺寸且形状规则的区域，如 2×2、3×3 的正方形，2×3、3×4 的矩形，或用来近似表示圆及椭圆等形状的多边形等。

邻域操作包括滑动邻域操作和分离邻域操作两种，滑动邻域是一个像素集，像素集包括的元素由中心像素的位置决定。滑动邻域操作一次处理一个像素，即对于输入图形的每一个像素，指定的操作将决定输出图像对应的像素值。图 4.15 说明了一个 6×5 矩形中三个像素的 2×3 滑动邻域，每一个邻域的中心像素都用一个黑点标出。

如果邻域含有奇数行和列，那么中心像素就是邻域的中心；如果行和列中有一维为偶数，那么中心像素将位于中心偏左或偏上方，即对于 $m\times n$ 的邻域，其中心像素的坐标为：

```
floor(([m n]+1)/2)
```

如果是 2×2 的邻域，其中心像素就是[1 1]，再如图 4.15 中 2×3 的邻域，其中心像素的坐标即为[1,2]。

实现一个滑动邻域操作需要以下几个步骤：

(1) 选择一个单独的像素；

(2) 确定该像素的滑动邻域;

(3) 对邻域中的像素值应用一个函数求值,该函数将返回标量计算结果;

(4) 将计算结果作为输出图像中对应的像素的值;

(5) 对输入图像的每一个像素都重复上面四个步骤。

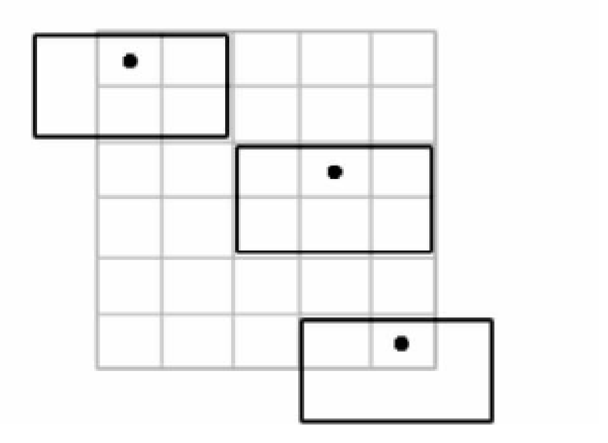


图 4.15 滑动邻域示意图

在进行滑动邻域操作时,邻域内某些像素可能会丢失,例如,6×5 矩形中左上角和右下角的两个邻域均含有不属于原图像的像素,因此 Matlab 在处理这些邻域时会在整个图像的矩形区域外部用若干全零行和列进行填充,而这些行和列不会成为输出图像的一部分,仅仅是在运算时作为临时值出现。

Matlab 图像处理工具箱提供了 nlfilter 函数进行滑动邻域操作,其使用格式如下:

```
Y=nlfilter(X,'index',[m n],fun,P1,P2,...)
```

其中 X 表示原图像,[m n]指定了大小为 $m\times n$ 的滑动邻域,fun 是对 $m\times n$ 的矩阵进行操作的函数,Pi 是传递给 fun 的附加参数。index 是可选参数,如果指定这个参数则表示函数将把原图像作为索引图像进行处理。

下面这条指令对原图像 I 的每个 3×3 的邻域进行标准方差的运算,结果返回给 I2。

```
I2=nlfilter(I,[3 3],'std2');
```

nlfilter 函数的 fun 参数可以是函数句柄,即用户可以编写 M 文件来实现自己所需要的计算函数,然后将该函数作为 nlfilter 的参数,下面的这行代码中用户编写的 myfun.m 文件可以被 nlfilter 函数所调用。

```
B = nlfilter(A,[3 3],@myfun);
```

下例使用 nlfilter 函数将每一个像素设置为 3×3 邻域内的最大值。

```
I=imread('tire.tif');
f=inline('max(x(:))');
I2=nlfilter(I,[3 3],f);
imshow(I);
figure, imshow(I2);
```

这段程序在 Matlab 中执行后的显示结果如图 4.16 所示。

Matlab 还提供了一种用于快速邻域操作的函数 colfilt,这个函数为每个像素都建立一个列向量,向量的各元素对应该像素的邻域的元素,即原始图像中的每一个像素都对应于 colfilt 函数所创建的临时矩阵的一个单独列。图 4.17 说明了临

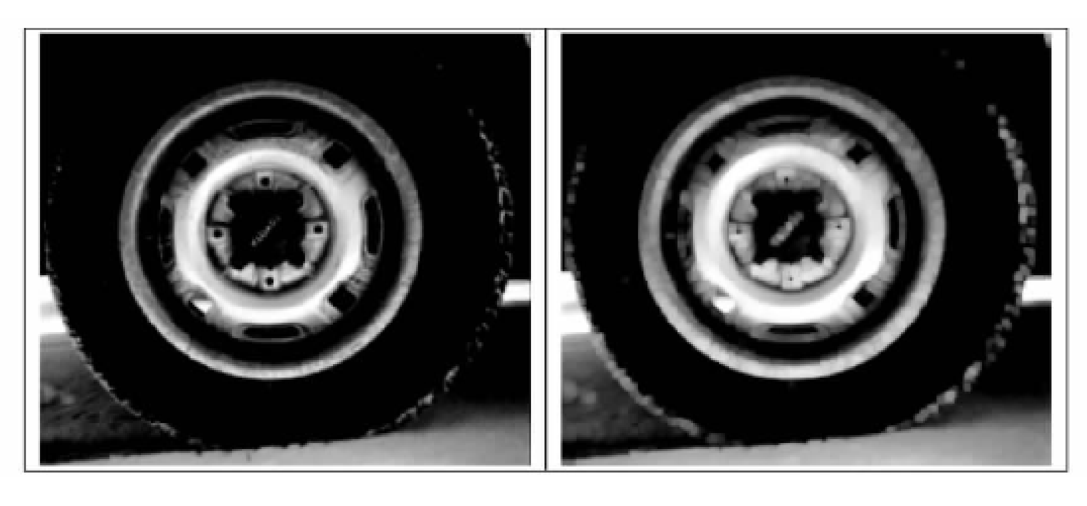


图 4.16　nlfilter 函数使用示例

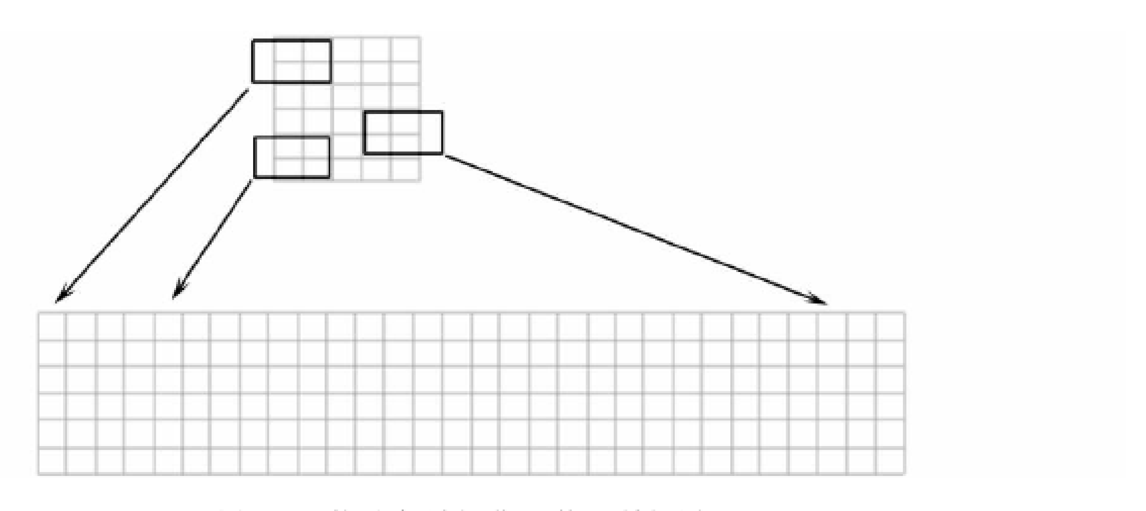


图 4.17 快速邻域操作函数 colfilt 原理

时矩阵列与图像像素之间的对应关系，其中 6×5 的矩阵以 2×3 的邻域为单位进行处理。

colfilt 函数为原始矩阵建立了一个 6×30 的临时矩阵，这个函数还可以根据需要对原图像进行补 0。

colfilt 函数的使用格式如下：

```
Y=colfilt(X,[m n],block_type,fun)
Y=colfilt(X,[m n],block_type,fun,P1,P2,...)
Y=colfilt(X,[m n],[mblock nblock],block_type,fun,...)
Y=colfilt(X,'indexed',...)
```

可以看出，这个函数的用法与 nlfilter 函数相似，X 表示原图像，[m n]指定了大小为 $m\times n$ 的滑动邻域，fun 是对 $m\times n$ 的矩阵进行操作的函数，Pi 是传递给 fun 的附加参数。indexed 是一个可选参数，如果指定这个参数则表示函数将把原图像作为索引图像进行处理。

block_type 参数表示块的移动方式，其中有两个取值：distinct 和 sliding，前者表示分离块操作，而后者表示图像的滑动块操作。[mblock nblock]则表示图像块

的大小为 $m\times n$。下例是将 5×5 的邻域内的所有像素值设为该邻域的平均值。

```
I=imread('tire.tif')
imshow(I)
I2=uint8(colfilt(I,[5 5],'sliding',@mean));
figure, imshow(I2)
```

这段程序在 Matlab 中执行后的显示结果如图 4.18 所示




图 4.18 colfilt 函数使用示例

在上例中,colfilt 函数生成的临时矩阵被传递给 fun 参数,实际上这个参数还可以是自定义的函数,它为矩阵的每一列返回一个单独的值,这个返回值将赋给输出图像中对应的像素。下面两行代码就是自定义函数的一个例子:

```
func=inline('min(x)');
J=colfilt(I , [6 6] , 'sliding' , func );
```

在本节前面介绍 nlfilter 函数时,类似的用法是 inline 函数的参数 max(x(:)),而不是这里的 min(x),因为 colfilt 函数操作中每个像素的邻域都已排列为列向量。

4.3.2 图像的块操作

在 Matlab 中,图像的分离块操作是将图像划分为大小相同的矩形区域,不同图像块在图像中无重叠地排列,其顺序从左上角开始,不足的部分可以在右下角补 0。如图 4.19 所示,一个 15×30 的矩阵被划分为 9 个大小为 4×8 的邻域,然后在底部和右部补 0,最后矩阵大小变为 16×32。

在 Matlab 图像处理工具箱中,对应分离块操作的函数是 blkproc,其调用格式如下:

```
Y=blkproc(X,[m n],fun)
Y=blkproc(X,[m n],fun,P1,P2,...)
Y=blkproc(X,[m n],[mborder nborder],fun,...)
Y=blkproc(X,'indexed',...)
```

这个函数基本上和滑动邻域操作函数相同,参数[mborder nborder]指定了图

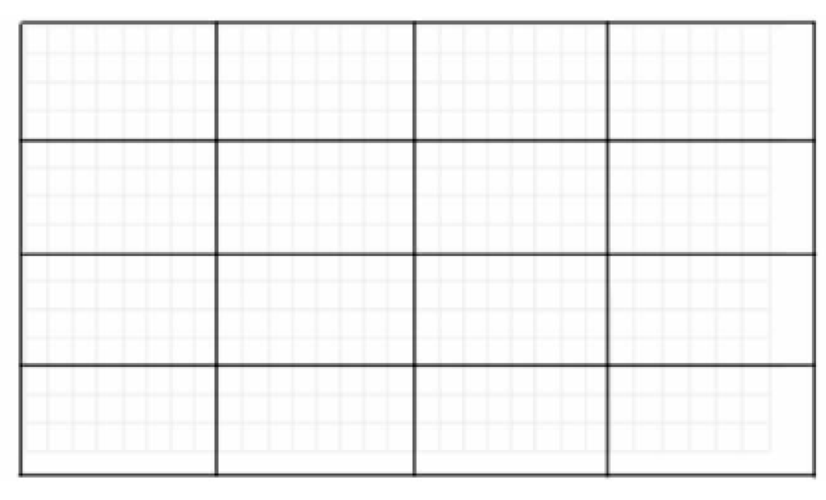


图 4.19　图像的块操作示意图

像块的扩展边界。即经过扩展的图像块的实际大小为($m+2\times$mborder)*($n+2\times$nborder),同样 blkproc 函数也会在必要时进行补 0,最后的操作将在扩展以后的块上进行。

下面这行代码中指定分离块的大小为 4×6,同时其行边界为 2,列边界为 3,因此实际上 fun 参数操作的块大小为 8×12。

```
B=blkproc(A,[4 6],[2 3],fun,...)
```

同样,fun 参数可以指定为以符号@开头的函数句柄,下例中即是将二维 dct 函数作为句柄传递给 blkproc 函数。

```
I=imread('cameraman.tif');
fun=@dct2;
J=blkproc(I,[8 8],fun);
imagesc(J), colormap(hot)
```

这段程序在 Matlab 中执行后的显示结果如图 4.20 所示。

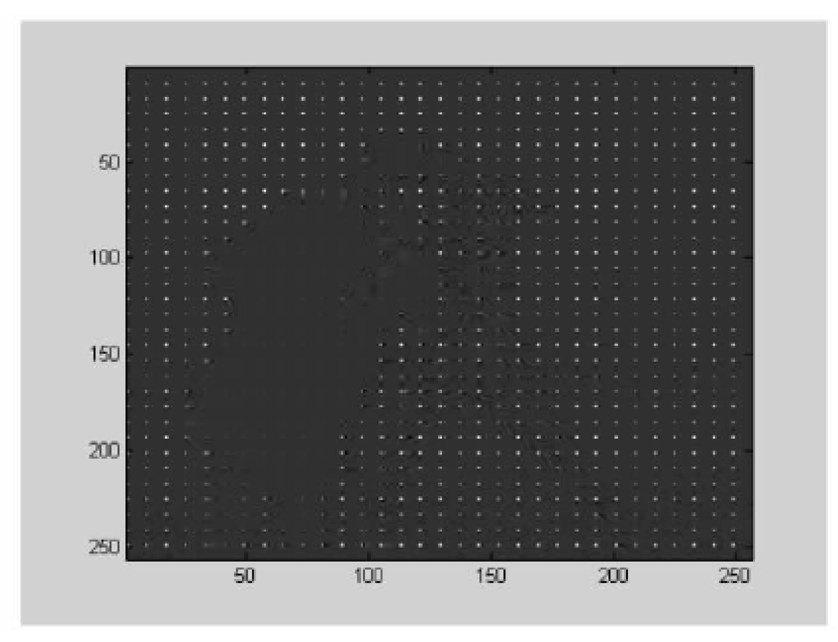


图 4.20　blkproc 函数使用示例

fun 参数同样也可以使用一个内联对象。下例是计算图像 8×8 区域的局部标准差。

```
I=imread('alumgrns.tif');
fun=inline('std2(s) * ones(size(x))');
I2=blkproc(I,[8 8],'std2(x) * ones(size(x))');
imshow(I)
figure, imshow(I2,[]);
```

这段程序在 Matlab 中执行后的显示结果如图 4.21 所示。

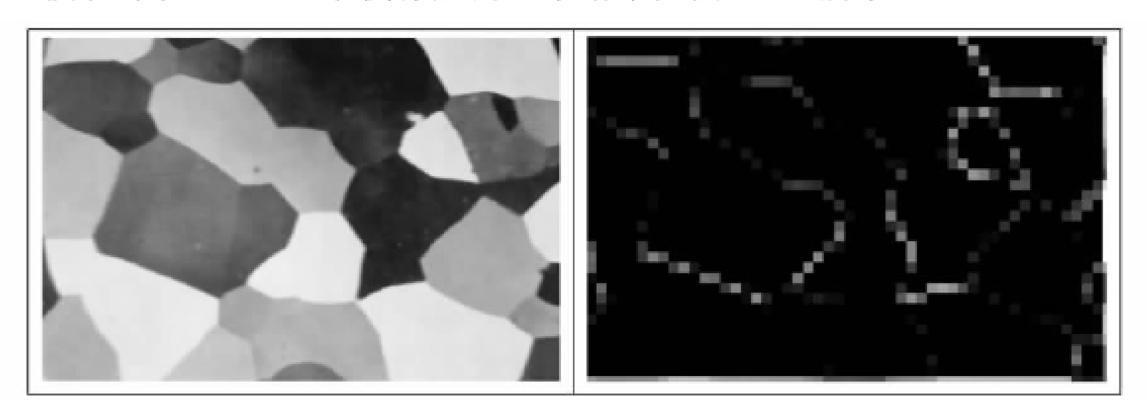


图 4.21　计算图像 8*8 区域的局部标准差

调用 blkproc 定义分离块时也可以使块之间互相重叠,在进行邻域处理时要重点考虑重叠部分的像素值。图 4.22 说明了一些在 15×30 矩阵中有 1×2 重叠部分的块,图中用阴影表示出来:

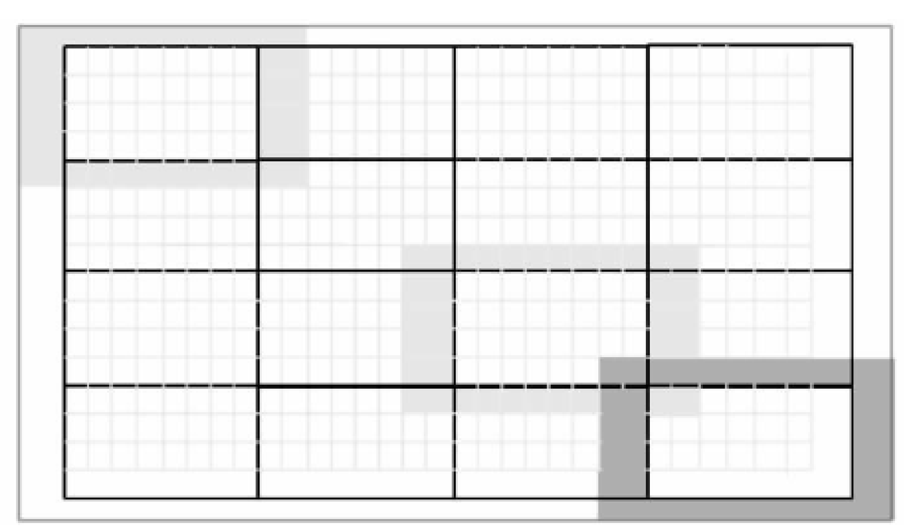


图 4.22　调用 blkproc 定义分离块

如果存在重叠现象,blkproc 函数则将扩展后的邻域(包括重叠部分)传递给指定的函数。此时 blkproc 函数需要一个额外的参数来指定重叠部分的大小,如:

```
B=blkproc(A,[4 8],[1 2],@myfun)
```

通常重叠操作会增加补 0 的数目,例如,在前面提到的 15×30 的矩阵经过填充后变成 16×32 的矩阵,如果有 1×2 的重叠部分,那么填充后的矩阵则会变为 18×36。

使用 colfilt 函数应用于分离块操作时，会将输入图像的每一个块进行重新排列来创建一个临时矩阵，在创建这个矩阵之前也需要进行补 0 操作。如图 4.23 所示，使用 4×6 的邻域对 6×6 的矩阵进行处理，colfilt 函数首先对图像进行 0 填充，使之成为一个 8×18 的矩阵，然后将块重新排列为 6 列，每列包含 24 个元素。也就是说，如果块的大小为 $m \times n$，而图像的大小为 $mm \times nn$，则临时矩阵的大小为 $(m \times n) \times (\text{ceil}(mm/m) \times \text{ceil}(nn/n))$。

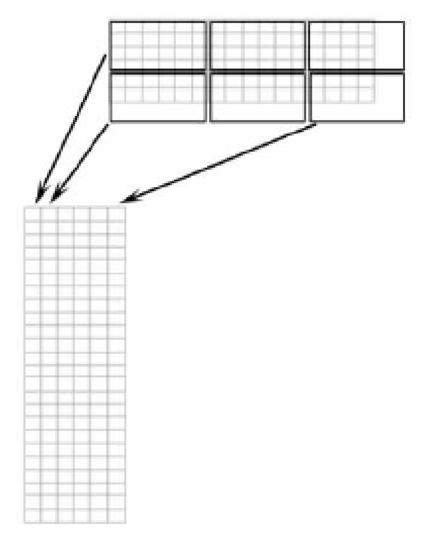


图 4.23　使用 colfilt 函数应用于分离块操作的原理说明

下例说明了使用 colfilt 函数进行分离块操作可以得到和 blkproc 函数相同的结果：

```
I=im2double(imread('tire.tif'));
f=inline('ones(64,1) * mean(x)');
I2=colfilt(I,[8 8],'distinct',f);
```

下面再介绍几个与块操作相关的工具函数。

（1）bestblk 函数。该函数用于选择图像块的大小，其使用格式为：

```
siz=bestblk([m n],k)
[mb,nb]=bestblk([m n],k)
```

第一行代码返回对 $m \times n$ 的图像的块划分大小，k 为图像块长度和宽度的最大值。第二行代码返回 mb 和 nb，分别为图像块的行数和列数。

例如：

```
siz=bestblk([640 800],72)


siz=
  64   50


col2im
```

（2）col2im 函数。该函数用于将向量重新排列为图像块，以发挥 Matlab 向量

计算能力强的特点,但在处理完成之后需要将向量重新排列为矩阵。其使用格式如下:

```
A=col2im(B,[m n],[mm nn], block_type)
A=col2im(B,[m n],[mm nn])
```

其中 B 为原图像,A 是将 B 的每一列重新排列成 $m \times n$ 的图像块之后组合为 $mm \times nn$ 的新的图像,最后一个参数 block_type 的含义和前面介绍的相同,包括 distinct 和 sliding 两个值,其默认值为 sliding。

(3) im2col 函数。该函数用于将图像块排列成向量,其使用格式如下:

```
B=im2col(A,[m n],block_type)
B=im2col(A,[m n])
B=im2col(A,'indexed',...)
```

该函数是将图像 A 的每一个 $m \times n$ 的块转换为一列,然后重新组合为 B。参数 block_type 的含义和默认值与 col2im 函数完全一致。

第 5 章　Matlab 图像变换

在数字图像处理中,把数字图像处理的方法主要分成两大类:一是空域分析法,二是频域分析法。空域分析法就是对图像矩阵进行处理;频域分析法则是通过图像变换将图像从空域变换到频域,再从另外一个角度来分析图像的特征并进行处理。频域分析法在图像增强、图像复原、图像编码压缩以及特征编码压缩方面有着广泛的应用。

图像变换就是把数字图像从空域变换到频域,即对原图像函数寻找一个合适的变换的数学问题。众多的图像变换方法不断涌现,从古老的傅里叶变换发展到余弦变换再到现在风靡世界的小波变换。所有的变换虽然名称各不相同,但是有一点是相同的,即每一个变换都存在自己的正交变换集,正是由于各种正交函数集的不同而形成了不同的变换。就像表示空间的一个向量可以用不同的坐标系表示一样,变换的途径虽然不同,但是他们都是空间域图像的变换域表示式。

在 Matlab 的图像处理工具箱中提供了一些对图像进行变换的函数,本章将一一介绍。

5.1　傅里叶变换

傅里叶变换是线性系统分析的一个有力工具,它将图像从空域变换到频域,从而很容易地了解到图像的各空间频域成分,从而进行相应的处理。傅里叶变换应用十分广泛,如图像特征提取、空间频率域滤波、图像恢复、纹理分析等。

5.1.1　傅里叶变换基础

1. 一维连续傅里叶变换

假设函数 $f(x)$ 为实变量 x 的连续函数,且在 $(-\infty,+\infty)$ 内绝对可积分,则 $f(x)$ 的傅里叶变换定义如下:

$$F(u)=\int_{-\infty}^{+\infty} f(x)\mathrm{e}^{-\mathrm{j}2\pi ux}\mathrm{d}x \tag{5.1}$$

其反变换为:

$$f(x) = \int_{-\infty}^{+\infty} F(u) e^{j2\pi ux} du \tag{5.2}$$

在这里，$j^2=-1$。在积分区间，$f(x)$必须满足只有有限个间断点、有限个极值和绝对可积的条件，并且 $F(u)$也是可积的。

以上两式称为傅里叶变换对。正、反傅里叶变换的唯一区别是幂的符号，并且是可逆的。$F(u)$是一个复函数，它由实部和虚部构成。

$$F(u) = R(u) + jI(u) \tag{5.3}$$

$$|F(u)| = \sqrt{R^2(u) + I^2(u)} \tag{5.4}$$

$$\theta(u) = \arctan\left[\frac{I(u)}{R(u)}\right] \tag{5.5}$$

$|F(u)|$称为 $f(x)$的振幅谱或傅里叶谱。$F(u)$称为 $f(x)$的幅值谱，$\theta(u)$称为 $f(x)$的相位谱。$E(u)=F^2(u)$，$E(u)$称为 $f(x)$的能量谱。

2. 二维连续傅里叶变换

从一维傅里叶变换可以很容易地推广到二维傅里叶变换。

如果 $f(x,y)$连续可积，并且 $F(u,v)$可积，则存在以下傅立叶变换对，其中 u、v 为频率变量：

$$F(u,v) = \int_{-\infty}^{+\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy \tag{5.6}$$

$$f(x,y) = \int_{-\infty}^{+\infty} f(x,y) e^{j2\pi(ux+vy)} dudv \tag{5.7}$$

与一维傅里叶变换一样，二维傅里叶变换可以写为如下的形式：

$$F(u,v) = R(u,v) + jI(u,v) \tag{5.8}$$

振幅谱为：

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)} \tag{5.9}$$

相位谱为：

$$\theta(u,v) = \arctan\left[\frac{I(u,v)}{R(u,v)}\right] \tag{5.10}$$

能量谱为：

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{5.11}$$

幅值谱表明了各正弦分量出现的多少，而相位谱信息表明了各正弦分量在图像中出现的位置。对于整幅图像来说，只要各正弦分量保持原位，幅值就显得不那么重要。所以大多数实用滤波器都只影响幅值，而几乎不改变相位信息。

5.1.2 离散傅里叶变换

由于在计算机中图像的存储使用的是数字形式，连续的傅里叶变换不适用于计算机的处理，所以在计算机中傅里叶变换一般都采用离散傅里叶变换（DFT）。

在计算机中使用离散傅里叶变换主要是基于以下原因：

(1) 离散傅里叶变换的输入输出都是离散值，方便计算机的运算操作；

(2) 采用离散傅里叶变换，可以用快速傅里叶变换来实现，提高了运算速度。

离散傅里叶变换的定义为：假设 m 的离散值是 $0,1,2,\cdots,N-1$，得到离散化的函数值 $\{f(m_0),f(m_0+Dm),f(m_0+2Dm),\cdots, f(m_0+[N-1]Dm)\}$，表示相对于连续函数的任意 N 个统一的空间采样。

定义一维离散傅里叶变换的正反公式对为：

$$F(p)=\frac{1}{N}\sum_{m=0}^{N-1}f(m)\mathrm{e}^{-\mathrm{j}\pi pm/N} \quad p=0,1,2,\cdots,N-1 \tag{5.12}$$

$$f(m)=\sum_{p=0}^{N-1}F(p)\mathrm{e}^{-\mathrm{j}\pi pm/N} \quad m=0,1,2,\cdots,N-1 \tag{5.13}$$

假设 $f(m,n)$ 是一个离散空间中的二维函数则二维离散傅里叶变换的正反公式对为：

$$F(p,q)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}f(m,n)\mathrm{e}^{-\mathrm{j}(2\pi/M)pm}\mathrm{e}^{-\mathrm{j}(2\pi/N)qn} \quad \begin{array}{l}p=0,1,2,\cdots,M-1\\ q=0,1,2,\cdots,N-1\end{array} \tag{5.14}$$

$$f(m,n)=\frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}F(p,q)\mathrm{e}^{\mathrm{j}(2\pi/M)pm}\mathrm{e}^{\mathrm{j}(2\pi/N)qn} \quad \begin{array}{l}m=0,1,2,\cdots,M-1\\ n=0,1,2,\cdots,N-1\end{array} \tag{5.15}$$

$F(p,q)$ 称为 $f(m,n)$ 的离散傅里叶变换系数。

$F(0,0)$ 被称为函数傅里叶变换的 DC 分量(DC 表示直流，指电压恒定的电源，与电压值呈正弦变化的电源不同)。在 Matlab 中，矩阵的取值是从 1 开始，而不是从 0 开始，因此，在 Matlab 中 $f(1,1)$ 和 $F(1,1)$ 分别代表的是 $f(0,0)$ 和 $F(0,0)$。

二维离散傅里叶变换也可以定义为：

$$f(\omega_1,\omega_2)=\sum_{-\infty}^{+\infty}\sum_{-\infty}^{+\infty}f(m,n)\mathrm{e}^{-\mathrm{j}\omega_1 m}\mathrm{e}^{-\mathrm{j}\omega_2 m} \tag{5.16}$$

$$f(m,n)=\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}F(\omega_1,\omega_2)\mathrm{e}^{-\mathrm{j}\omega_1,m}\mathrm{e}^{-\mathrm{j}\omega_2 m}\mathrm{d}\omega_1\mathrm{d}\omega_2 \tag{5.17}$$

式中：ω_1 和 ω_2 为频域变量(rad/采样单元)。

【例 5.1】如图 5.1 所示，函数 $f(m,n)$ 在一个矩形区域内函数值为 1，而在其他区域为 0。将这个函数的幅值显示为网格图。

为了简便，假设 $f(m,n)$ 是一个连续函数，图 5.2 显示了函数的傅里叶变换幅值。在图中心 $F(0,0)$ 的顶峰位置处的取值是函数 $f(m,n)$ 所有取值的和。从图 5.2 中还可以看到在 $f(\omega_1,\omega_2)$ 变换中，高频垂直频率的能量比高频水平频率的能量多，这反映函数傅里叶变换的相似性：$f(m,n)$ 变换在水平方向上是窄带脉冲，在垂直方向是宽带脉冲，窄带脉冲的高频分量比宽带脉冲多。

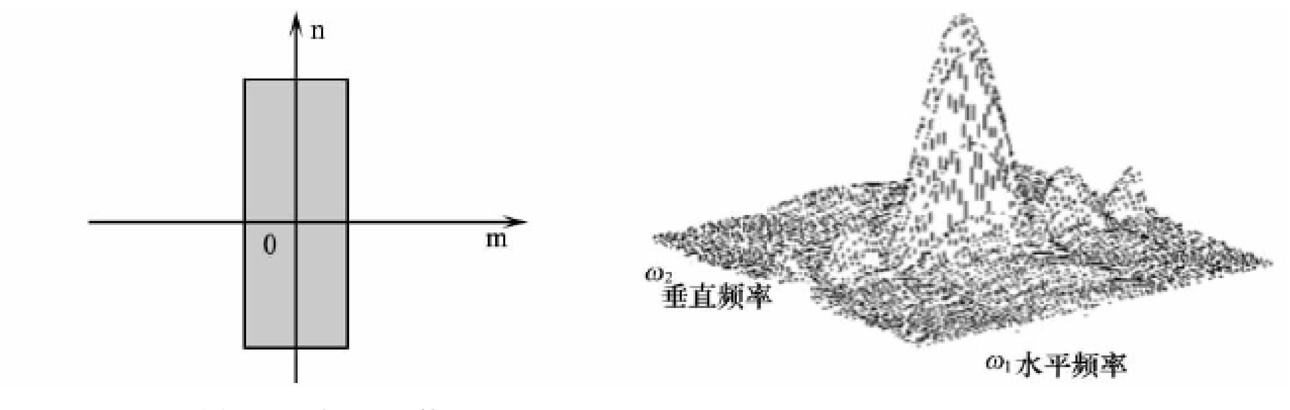


图 5.1　矩形函数

图 5.2　函数 $f(m,n)$傅里叶变换幅值的网格显示

将傅里叶函数可视化的另一种方法是将 $\lg|F(\omega_1,\omega_2)|$ 显示为一幅图像。上例中用这种方法得到的显示结果如图 5.3 所示。

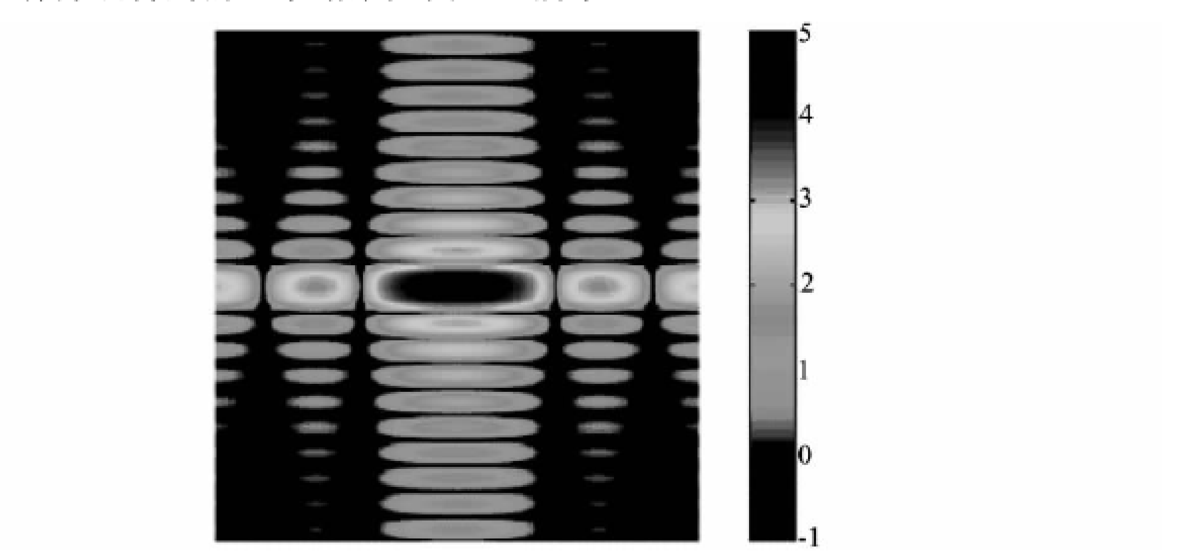


图 5.3　对数显示傅里叶变换幅值

用这种方法可以突出 $f(\omega_1,\omega_2)$傅里叶子变换在靠近 0 处的细节。

下面通过一个例子来看如何实现图像矩阵的傅里叶变换。

【例 5.2】绘制一个二值图像矩阵,并将其傅里叶函数可视化。

说明:这个例子将创建一个类似于上例的 $f(m,n)$的矩阵,然后用一个二进制图像显示,再对这个图像矩阵做傅里叶变换,显示幅值。

```
%创建矩阵
f = zeros(40,40);
f(5:24,13:17) = 1;
subplot(1,2,1)
imshow(f);title('矩阵的显示');
%进行图像 f 的傅里叶变换
F=fft2(f);
%对幅值取对数
F2=log(abs(F));
```

```
subplot(1,2,2)
imshow(F2,[-1 5],'notruesize');title('傅里叶变换显示');
```

其显示结果如图 5.4 所示。

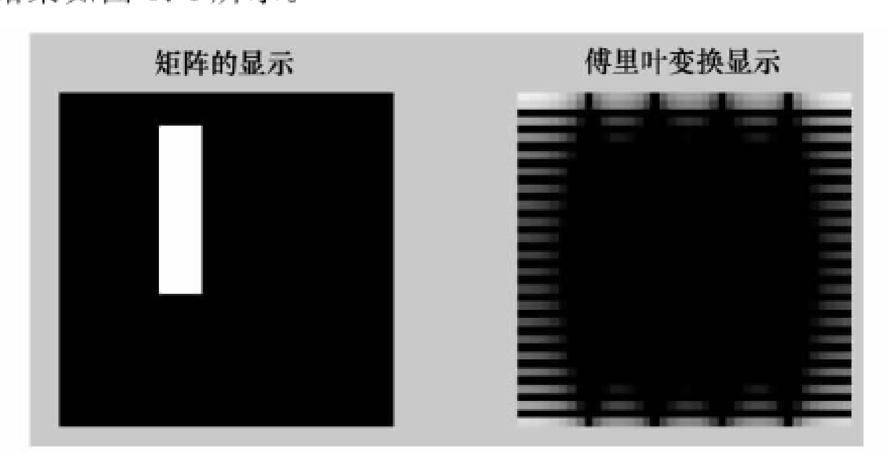


图 5.4　傅里叶变换显示

几种常见函数傅里叶变换的幅值显示如图 5.5 所示。

(a)

(b)

(c)

图 5.5　几种常见函数傅里叶变换的幅值

二维傅里叶变换具有很多重要性质，如可分离性、平移性、周期性、共轭对称性、旋转性、满足分配律、可进行尺度变换(缩放)、卷积等。这些性质使得在频域中

对图像的操作变得简单，从而简化了整个处理过程，因此在图像处理中都有重要的应用。表 5.1 列出了其中的一些性质。

表 5.1 傅里叶变换的性质

性 质	空 域	频 域
加法定理	$f(x,y)+g(x,y)$	$F(u,v)+G(u,v)$
相似性定理	$f(ax,by)$	$\frac{1}{\lvert ab \rvert}F(\frac{u}{a},\frac{v}{b})$
位移定理	$f(x-a,y-b)$	$e^{-j2\pi(au+bv)}F(u,v)$
卷积定理	$f(x,y)\times g(x,y)$	$F(u,v)G(u,v)$
可分离乘积	$f(x)g(y)$	$F(u)G(v)$
微分	$(\frac{\partial}{\partial x})^m(\frac{\partial}{\partial y})^n f(x,y)$	$(j2\pi u)^m(j2\pi v)^n F(u,v)$
旋转	$f(x\cos\theta+y\sin\theta,-x\sin\theta+y\cos\theta)$	$F(u\cos\theta+v\sin\theta,-u\sin\theta+v\cos\theta)$
拉普拉斯变换	$\nabla^2 f(x,y)=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}f(x,y)$	$-4\pi^2(u^2+v^2)F(u,v)$
Rayleigh 定理	$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\lvert f(x,y)\rvert^2 dxdy=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\lvert F(u,v)\rvert^2 dudv$	

5.1.3 快速傅里叶变换

离散一维傅里叶变换运算的乘法和加法运算的数量级是 N^2，这样的运算量太大。而快速傅里叶变换(FFT)将 DFT 计算分解，将运算次数降为 $N\log_2 N$。

其算法思想是：

(1) 将原图像进行转置；

(2) 按行对转置后的图像矩阵做一维 FFT，将这变换后的中间矩阵再转置；

(3) 对转置后的中间矩阵做一维 FFT，最后得到的就是二维 FFT。

快速傅里叶算法只能处理大小为 2 的幂次的矩阵(其他大小的矩阵可以采用其他非基 2 的混合基算法)。

【例 5.3】对矩阵进行零填充后，进行快速傅里叶变换。

```
f=zeros(30,30);
f(5:24,13:17)=1;
%进行零填充后，进行快速傅里叶变换
F=fft2(f,256,256);
F2=log(abs(F));
imshow(F2,[-1 5],'notruesize'); colormap
(jet); colorbar
```

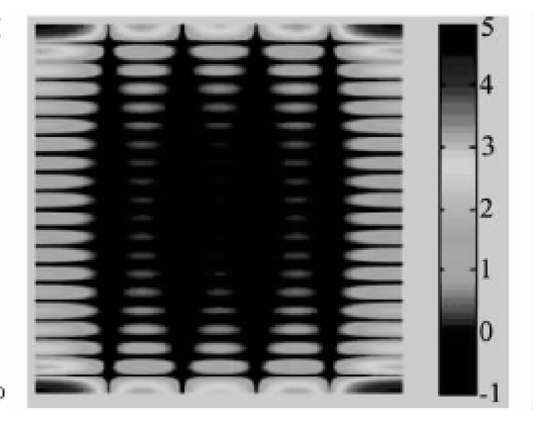


图 5.6 快速傅里叶变换显示

其显示结果如图 5.6 所示。

5.1.4 傅里叶变换的应用

Matlab 的图像处理工具箱提供了一些函数
来进行傅里叶变换。

1．函数 fft2

功能：用于计算二维快速傅里叶变换。

语法格式：

```
Y=fft2(X)
Y=fft2(X,m,n)
```

说明：

X 是输入图像矩阵，***Y*** 是返回图像矩阵 ***X*** 的二维傅里叶变换，***X*** 和 ***Y*** 大小相同。

在 Y=fft2(X,m,n)中，按照 m、n 指定的值对图像进行剪切或补零后进行傅里叶变换，返回变换矩阵的大小为 $m\times n$。

2．函数 fftn

功能：用于计算 n 维傅里叶变换。

语法格式：

```
Y=fftn(X)
Y=fftn(X,size)
```

说明：

X 是输入图像矩阵，***Y*** 是返回图像矩阵 ***X*** 的 n 维傅立叶变换，***X*** 和 ***Y*** 大小相同。

在 Y=fftn(X,size)中，按照 size 指定的值对图像进行剪切或补零后进行傅里叶变换，返回变换矩阵的大小为 size。

3．函数 fftshift

功能：将傅里叶变换后的图像频谱中心从矩阵的原点移动到矩阵的中心。

语法格式：

```
Y=fftshift(X)
Y=fftshift(X,dim)
```

说明：

ffishift 用来调整 fft、fft2 和 fftn 的输出结果。对于向量 ***X***，将其左右两半交换维值；对于矩阵 ***X***，将其一、三象限和二、四象限进行互换；对于高维向量，将矩阵各维的两半进行互换。

4．函数 ifft2

功能：计算二维傅里叶变换的反变换。

语法格式：

```
Y=ifft2(X)
Y=ifft2(X,m,n)
```

说明：

参数定义同函数 fft2。

5. 函数 ifftn

功能：计算 n 维傅里叶变换的反变换。

语法格式：

```
Y=ifftn(X)
Y=ifftn(X,siz)
```

说明：

参数定义同函数 fftn。

傅里叶函数在 Matlab 图像处理中具有广泛的应用，下面介绍它的一些例子。

1. 线性过滤器的频率响应

由线性系统理论可知，线性过滤器脉冲响应的傅里叶变换给出了对该过滤器的频率响应。信号处理工具箱的 freqz2 函数就是计算滤波器的频率响应并且显示。

【例 5.4】求线性滤波器的频率响应。

```
h=fspecial('gaussian');
freqz2(h)
```

其显示结果如图 5.7 所示。

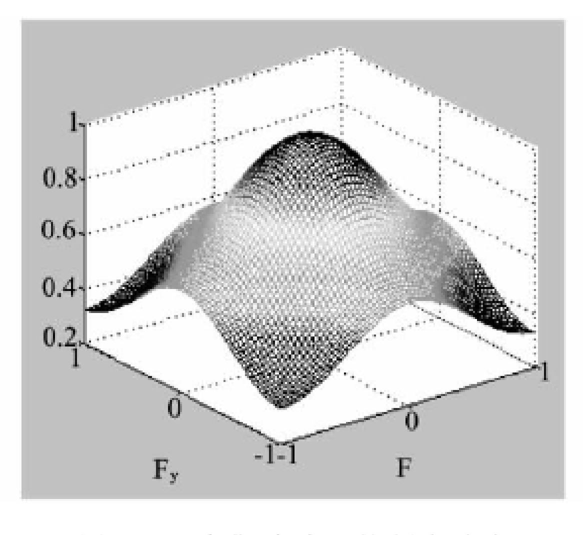


图 5.7　高斯滤波器的频率响应

2. 快速卷积

傅里叶变换的一个重要特征是两个傅里叶变换的乘积与相应空间的函数的卷积相对应。该特性与快速傅里叶变换一起，共同形成快速卷积算法的基础。

假设 $\boldsymbol{A}$ 为 $M\times N$ 的矩阵，$\boldsymbol{B}$ 为 $P\times Q$ 的矩阵，则可以通过下面的方法来计算 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的卷积。

算法：

(1) 对 **A** 和 **B** 进行 0 填充，将它们均达到至少$(M+P\cdot 1)\times(N+Q\cdot 1)$的矩阵；

(2) 使用 fft 函数计算 **A** 和 **B** 的二维 DFT；

(3) 将两个 DFT 结果相乘；

(4) 利用 ifft2 函数对上面得到的乘积进行 DFT 反变换。

【例 5.5】计算一个魔方阵和全 1 矩阵的卷积。

```
A = magic(3);      %用 1 到 9 之间的数产生一个 3×3 矩阵
B = ones(3);       %生成一个全 1 的 3×3 矩阵
% 对矩阵 A 进行 0 填充，得到一个 8×8 矩阵
A(8,8) = 0;
% 对矩阵 B 进行 0 填充，得到一个 8×8 矩阵
B(8,8) = 0;
%分别对 A 和 B 作傅里叶变换；将傅里叶变换的结果在频域内进行点乘，最后将点%
乘的结果变换回空域
C=ifft2(fft2(A).*fft2(B));
%截取有效数据
C=C(1:5,1:5);
C=real(C)
```

得到 A 与 B 的卷积 C 为：

```
C=
    8.0000    9.0000    15.0000    7.0000    6.0000
    11.0000   17.0000   30.0000    19.0000   13.0000
    15.0000   30.0000   45.0000    30.0000   15.0000
    7.0000    21.0000   30.0000    23.0000   9.0000
    4.0000    13.0000   15.0000    11.0000   2.0000
```

3. 对图像定位的应用

傅里叶变换可以用于与卷积密切相关的运算。在数字图像处理相关的运算中用于匹配模板，可以对某些模板对应的特征进行定位。算法是：以待定的目标为模板在待识别图像上滑动，同时联贯运算，对运算结果取适当的阈值，确定待定位的目标的位置。

【例 5.6】如图 5.8 所示，在图像 text.tif 中定位字母 a。

说明：将字母 a 和图像 text.tif 进行快速傅里叶变换，然后计算字母 a 和图像的卷积，最后计算卷积运算的峰值，就得到了在图像中对字母 a 的定位。

```
w=imread('text.tif');
%将字母 a 从图像中切割出来
a=bw(59:71,81:91);
imshow(bw);
figure;imshow(a);
```

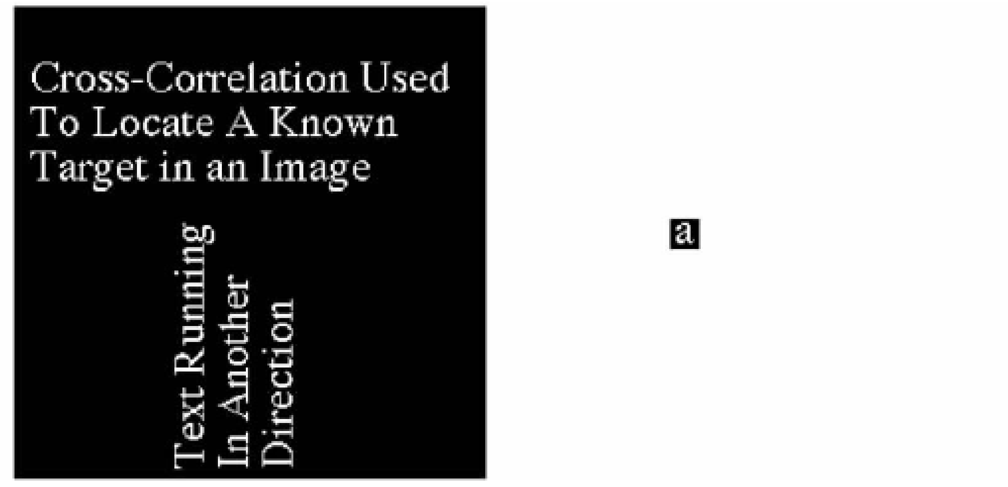


图 5.8　text.tif 和字母 a

```
%将字母 a 和图像 text.tif 进行快速傅里叶变换,然后计算字母 a 和图像的卷积
C = real(ifft2(fft2(bw) .* fft2(rot90(a,2),256,256)));
figure, imshow(C,[]);
max(C(:))                    %对卷积后的图像归一化
thresh = 45;                 %设置阈值
%显示像素值超过阈值的像素
figure, imshow(C > thresh)
```

其显示结果如图 5.9 所示,图 5.9(b)中的亮点是字母 a 出现的位置。

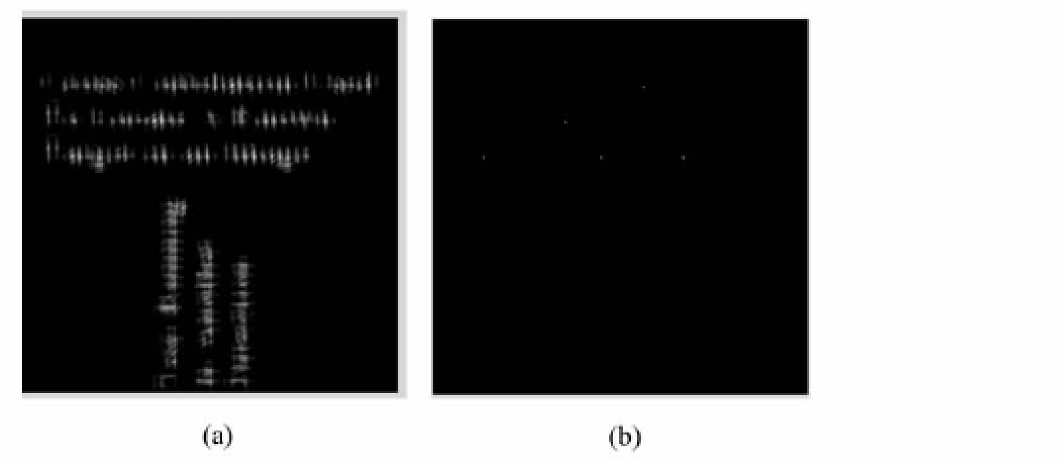


图 5.9　显示结果

(a)卷积运算的结果;(b)匹配到的字符位置。

5.2　离散余弦变换

5.2.1　离散余弦变换基础

在傅里叶级数展开式中,如果函数对称于原点,则其级数中将只有余弦函数项。从这一现象受到启示,人们提出了另一种图像变换方法——离散余弦变换(DCT)。由于离散余弦变换具有把图像的重要可视信息都集中在变换的一小部

分系数中，所以，DCT 在图像的压缩中非常有用，是 JPEG(Joint Photographic Expert Group)算法的基础。

一维离散余弦正反变换公式如下：

$$F(k) = 2\sum_{n=0}^{N-1} f(n)\cos\frac{\pi(2n+1)k}{2N} \qquad n,k = 0,1,\cdots,N-1 \tag{5.18}$$

$$f(n) = \frac{1}{N}\sum_{k=0}^{N-1} F(k)\cos\frac{\pi(2n+1)k}{2N} \qquad n,k = 0,1,\cdots,N-1 \tag{5.19}$$

类似于一维离散余弦变换，一个 $M\times N$ 矩阵 $\mathbf{A}$ 的二维离散余弦变换公式如下：

$$F(u,v) = c(u)c(v)\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{\pi(2x+1)u}{2M}\cos\frac{\pi(2y+1)v}{2N}$$

$$u = 0,1,\cdots,M-1$$
$$v = 0,1,\cdots,N-1 \tag{5.20}$$

二维离散余弦反变换公式如下：

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} c(u)c(v)F(u,v)\cos\frac{\pi(2x+1)u}{2M}\cos\frac{\pi(2y+1)v}{2N}$$

$$x = 0,1,\cdots,M-1$$
$$y = 0,1,\cdots,N-1 \tag{5.21}$$

其中：

$$c(u) = \begin{cases} \dfrac{1}{\sqrt{M}} & u = 0 \\ \sqrt{\dfrac{2}{M}} & u = 1,2,\cdots,M-1 \end{cases} \tag{5.22}$$

$$c(v) = \begin{cases} \dfrac{1}{\sqrt{N}} & v = 0 \\ \sqrt{\dfrac{2}{N}} & v = 1,2,\cdots,N-1 \end{cases} \tag{5.23}$$

在 Matlab 中，$c(u)c(v)F(u,v)\cos\frac{\pi(2x+1)u}{2M}\cos\frac{\pi(2y+1)v}{2N}$ 称为离散函数的基础函数。DCT 系数 $F(u,v)$ 可以看做应用于每一个函数的权值。

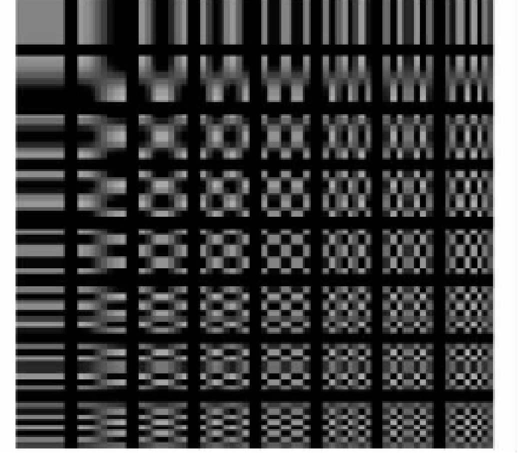


图 5.10　DCT 矩阵

例如，对于一个 8×8 矩阵，它的 64 个基本函数如图 5.10 所示。这些基本函数的水平频率从左到右增长，垂直频率从上到下增长。位于图像左上方的定值基本函数被称为 DC(余弦)基本函数，相应的 DCT 系数称为 DC 系数。

5.2.2 离散余弦变换的实现

离散余弦变换的实现有以下两种方法：

(1) 基于 FFT 的快速算法，工具箱中的 dct2 函数实现了这种算法；

(2) 基于 DCT 矩阵的算法，工具箱提供了 dctmtx 函数来计算变换矩阵。

一个 $M \times M$ 的 DCT 矩阵 $\boldsymbol{T}$ 定义为：

$$\boldsymbol{T} = (T_{i,j}) = \begin{cases} \dfrac{1}{\sqrt{M}} & i = 0,\ 0 \leqslant j \leqslant M-1 \\ \sqrt{\dfrac{2}{M}} \cos \dfrac{\pi(2j+1)i}{2M} & 1 \leqslant i \leqslant M-1,\ 0 \leqslant j \leqslant M-1 \end{cases} \tag{5.24}$$

则 $\boldsymbol{X}$ 的 DCT 变换 $\boldsymbol{Y}$ 为 $\boldsymbol{Y}=\boldsymbol{T}\times\boldsymbol{T}^{\mathrm{T}}$。

Matlab 图像处理工具箱提供了以下函数来进行离散余弦变换。

1. 函数 dct2

功能：实现图像的二维离散余弦变换。

语法格式：

```
B=dct2(A)
B=dct2(A,[M N])
B=dct2(A,M,N)
```

说明：

$\boldsymbol{A}$ 表示要变换的图像矩阵，$\boldsymbol{B}$ 表示变换后得到的图像矩阵。$\boldsymbol{B}$ 和 $\boldsymbol{A}$ 是同样大小的矩阵，并且包含了余弦变换后的系数。

M、N 是可选参数，表示对图像矩阵 $\boldsymbol{A}$ 的填充或截取。如果 M 或 N 小于矩阵 $\boldsymbol{A}$ 的对应维，则截取 $\boldsymbol{A}$；反之，当大于时，在变换前，用 0 把 $\boldsymbol{A}$ 填充为 $M \times N$ 矩阵。B=dct2(A,[M N])或者 B=dct2(A,M,N)在对 $\boldsymbol{A}$ 进行二维离散余弦变换前，先对 $\boldsymbol{A}$ 进行填充或截取。

优点：dct2 函数使用了一个基于 FFT 的算法提高了输入较大的矩阵时的计算速度。

【例 5.7】对图像 autumn.tif 进行离散余弦变换。

```
RGB=imread('autumn.tif');
%将彩色图像转换为灰度图像
I=rgb2gray(RGB);
subplot(1,2,1);
imshow(I);
title('原始图像');
%进行余弦变换
```

```
J=dct2(I);
subplot(1,2,2);
imshow(log(abs(J)),[]);
title('DCT 结果');
```

其显示结果如图 5.11 所示。



图 5.11 原始图像和变换的二维显示比较

可以看出图像的能量很大部分在变换矩阵的左上角。

2. 函数 idct2

功能:实现图像的二维离散余弦反变换。

语法格式:

```
B=idct2(A)
B=idct2(A,[M N])
B=idct2(A,M,N)
```

说明:

A 表示变换的二维离散余弦变换矩阵,**B** 表示原图像矩阵。**B** 和 **A** 是同样大小的矩阵,M、N 是可选参数,表示对二维离散余弦变换矩阵 **A** 的填充或截取。如果 M 或 N 小于矩阵 **A** 的对应维,则截取 **A**;反之,当大于时,在变换前,用 0 把 **A** 填充为 $M\times N$ 矩阵。B=idct2(A,[M N])或者 B=idct2(A,M,N)在对 **A** 进行二维离散余弦变换前,先对二维离散余弦变换矩阵进行填充或截取。

3. 函数 dctmtx

功能:计算二维 DCT 矩阵。

语法格式:

```
D=dctmtx(n)
```

说明:

D 是返回的 $n\times n$ 的 DCT 矩阵,如果矩阵 **A** 的大小是 $n\times n$,$\boldsymbol{D}\times\boldsymbol{A}$ 是矩阵 **A** 每一行的 DCT 值,$\boldsymbol{D}'\times\boldsymbol{A}$ 是每一列的 DCT 值。

如果矩阵 **A** 是 $n\times n$ 的方阵,则 **A** 的 DCT 可以用来进行 $\boldsymbol{D}\times\boldsymbol{A}\times\boldsymbol{D}'$ 计算。这在有些地方比 dct2 计算快,特别是对于 **A** 很大的情况。

【例 5.8】将例 5.7 中 DCT 后的矩阵中小于 10 的系数设为 0,然后重构图像。

```
J(abs(J)<10)=0;              %将小于 10 的变换系数设置为 0
K=idct2(J);                  %用 idct2 函数重建图像矩阵
figure;
imshow(K,[0 255])
title('重构后的图像');
```



图 5.12　余弦反变换恢复图像

下例是舍弃 DCT 变换矩阵的系数对图像质量的影响。

【例 5.9】对 DCT 变换矩阵舍弃系数后重构的图像和原图像进行比较。qtdemo 是 Matlab 自带的一个演示程序。输入 qtdemo 后,就会运行程序:
其显示结果如图 5.13 所示。

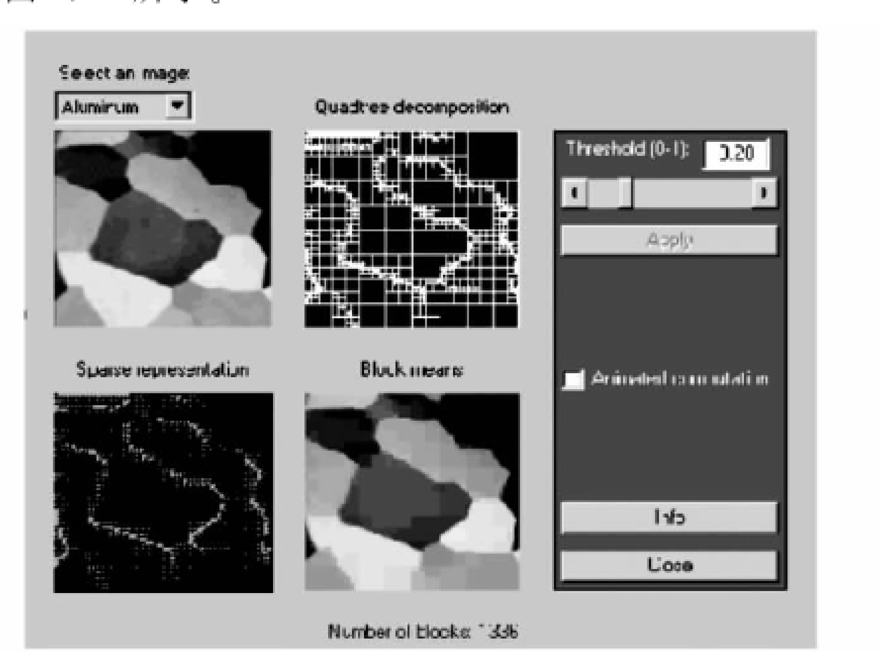


图 5.13　观察系数对重构图像影响的工具界面

这是一个观察 DCT 系数的舍弃对重构图像影响的例子。左上的是原始图像,左下的是舍弃系数后重构的图像,右下是重构图像和原始图像的差别。

在这个例子中图像被分为 8×8 的块来进行变换和重构,右上可以拖动滑条来

选择被舍弃的系数，黑色是被舍弃的 DCT 系数。可以看到舍弃 1/2 左右的系数都不会对画面质量造成明显的影响。

8×8 的图像块经过 DCT 后，其低频分量都集中在左上角，高频分量分布在右下角(DCT 实际上是空间域的低通滤波器)。由于该低频分量包含了图像的主要信息(如亮度)，而高频与之相比，就不那么重要了，所以可以忽略高频分量，从而达到压缩的目的。将高频分量去掉就要用到量化，它是产生信息损失的根源。这里的量化操作，就是将某一个值除以量化表中对应的值。由于量化表左上角的值较小，右上角的值较大，这样就起到了保持低频分量、抑制高频分量的目的。

5.3 Radon 变换

5.3.1 Radon 变换基础

图像的投影(Projection)就是图像在某一方向上的线性积分，对于数字图像来说也就是其在该方向的累加求和。

二维函数 $f(x,y)$ 的投影是其在确定方向上的线积分。例如，$f(x,y)$ 在垂直方向上的二维线性积分就是 $f(x,y)$ 在 x 坐标轴的投影；$f(x,y)$ 在水平方向上的二维线性积分就是 $f(x,y)$ 在 y 轴上的投影，如图 5.14 所示。

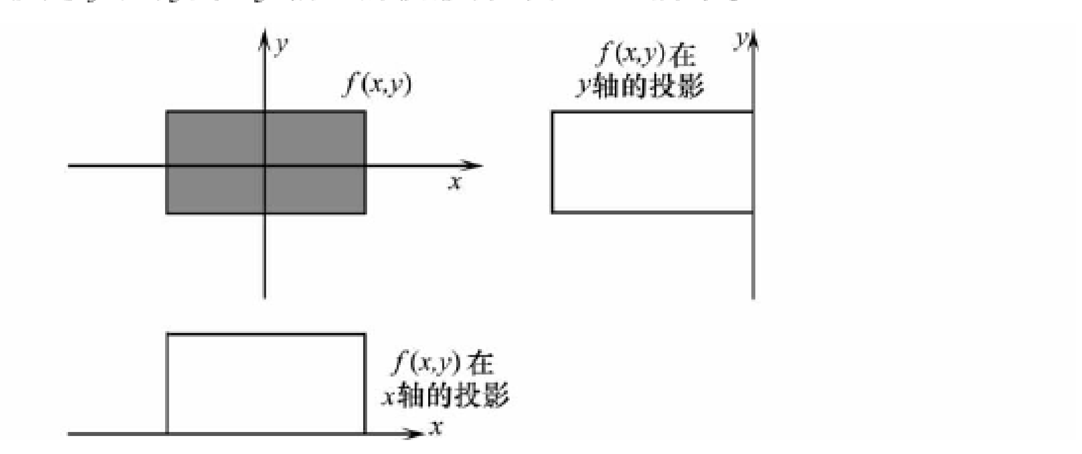


图 5.14　图像在 x 轴、y 轴的投影

所谓 Radon 变换，就是将计算机图像变换为在某一种指定角度射线方向上投影的变换方法。图 5.15 是 Radon 变换的几何关系示意图。

可以沿任意角度 θ 计算函数的投影，即任意角度上都存在函数的 Radon 变换。图像 $f(x,y)$ 在任一角度 θ 上的 Radon 变换的投影定义为：

$$R_\theta(x') = \int_{-\infty}^{+\infty} f(x'\cos\theta - y'\sin\theta, x'\sin\theta - y'\cos\theta)\mathrm{d}y' \qquad (5.25)$$

其中：

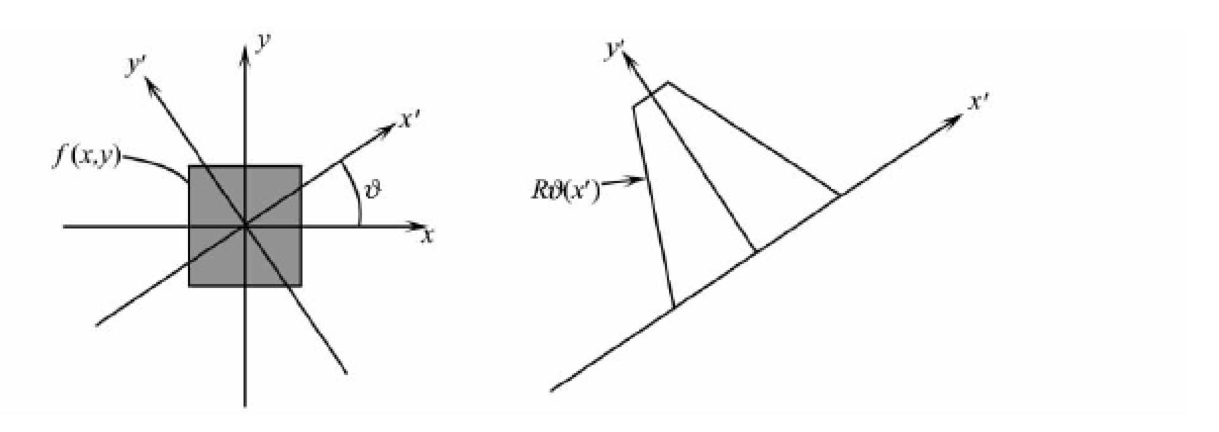


图 5.15 Radon 变换的几何关系示意图

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.26}$$

Matlab 图像处理工具箱提供了 radon 函数，用来计算图像在指定角度方向上的 Randon 变换。

其语法格式为：

```
[R xp]=radon(I,theta,N)
```

其中，I 表示需要变换的图像；R 的各行返回角度参数 theta 中各方向上的 Radon 变换值；***xp*** 向量表示沿 x' 轴相应的坐标值；N 是一个可选参数，指定 Radon 变换将在 N 点上进行计算，默认条件下投影点的数目由下式决定：

$$2*\mathrm{ceil}(\mathrm{norm}(\mathrm{size}(I)-\mathrm{floor}(\mathrm{size}(I)-1)/2-1))+3$$

如果指定了参数 N，那么 R 将包含 N 个行向量。

【例 5.10】计算方框图像（见图 5.16）在 0° 和 45° 方向上的 Radon 变换。

```
I=zeros(100,100);
%产生一个正方形的黑框
I(25:75, 25:75) = 1;
imshow(I);
%计算这个图像的 Radon 变换
[R,xp] = radon(I,[0 45]);
figure;
%显示图像在 0°方向上的 Radon 变换
plot(xp,R(:,1));
title('R_{0^o} (x\prime)')
figure;
%显示图像在 45°方向上的 Radon 变换
plot(xp,R(:,2));
title('R_{45^o} (x\prime)');
```

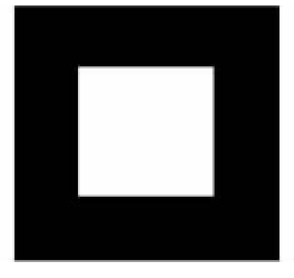


图 5.16 方框图像

其显示结果如图 5.17 所示。

有很多角度的 Radon 变换通常可以表示为一幅图像；大角度范围的 Radon 变

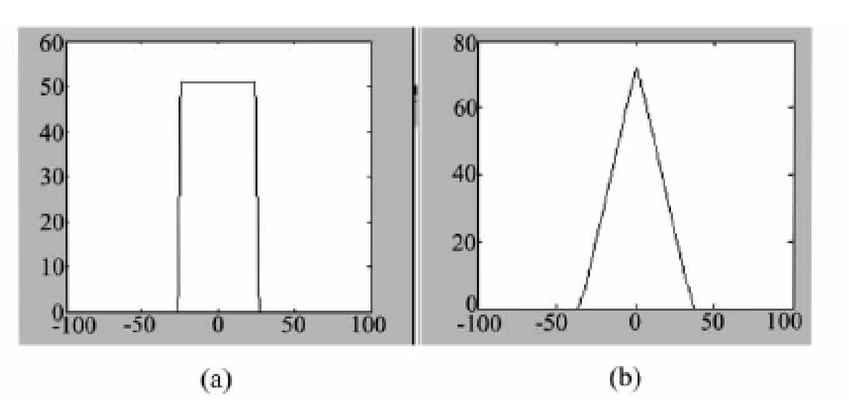


图 5.17 Radon 变换示例

(a)0°方向上的 Radon 变换；(b)45°方向上的 Radon 变换。

换通常都会显示为一幅图像。

【例 5.11】计算方框图像从 0°～180°每隔 1°增加的多个 Radon 变换。

```
I=zeros(100,100);             %生成一个全 0 矩阵
I(25:85, 25:85)=1;            %产生一个正方形黑框
theta = 0:180;
[R,xp]=radon(I,theta);
imagesc(theta,xp,R);title('R_{\theta} (X\prime)');
xlabel('\theta (degrees)');
ylabel('X\prime');
set(gca,'XTick',0:20:180);
colormap(hot);
colorbar
```

其显示结果如图 5.18 所示。

Radon 变换类似于计算机视觉中有名的 Hough 变换，可以用来检测图像中的直线。通常会综合使用这两种变换进行图像分析。例如，使用 Radon 函数实现某一种 Hough 变换，从而检测图像的直线。

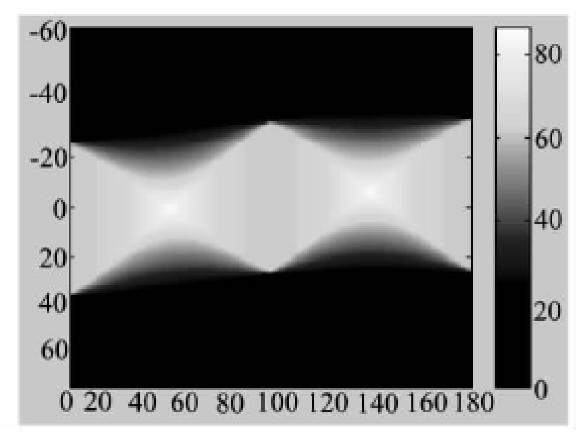


图 5.18 方框图像在 0°～180°的 Radon 变换

【例 5.12】用 Radon 变换检测图像中的直线。

实现方法如下：

(1) 用 edge 函数计算图像边缘的二值图像；

(2) 计算此边界图像的 Radon 变换；

(3) 计算出 Radon 变换矩阵的峰值，这些峰值的位置相对于原图像中的直线。

实现的程序代码如下：

```
%用 edge 函数计算出边界
```

```
I = imread('ic.tif');
BW = edge(I);
imshow(I)
figure, imshow(BW)
%进行 Radon 变换
theta = 0:179;
[R,xp] = radon(BW,theta);
figure, imagesc(theta, xp, R); colormap(hot);
%找出直线位置
xlabel('\theta (degrees)'); ylabel('X\prime');
title('R_{\theta} (X\prime)');
colorbar
```

其显示结果如图 5.19 所示。

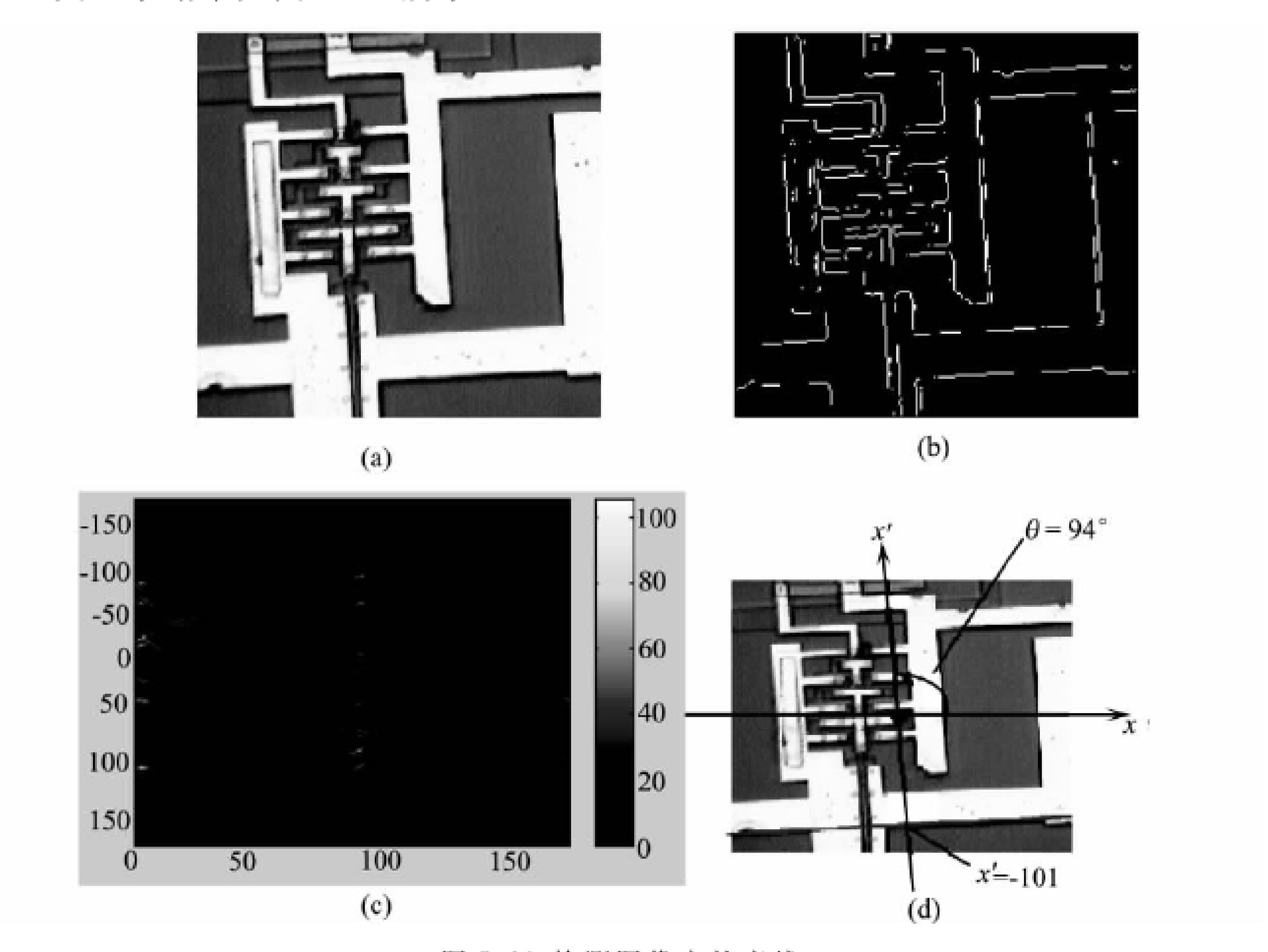


图 5.19 检测图像中的直线

(a)原始图像;(b)二进制边界图像;(c)边界图像的 Radon 变换;(d)原图像中的直线。

从图像中可以看到,Radon 变换矩阵在 $x'=-101$、$\theta=94°$处有一个峰值,则对应图像在 $\theta=94°$的坐标轴 x'的-101 处有一条垂直于 x'的直线。

5.3.2 Radon 逆变换

给定图像 I 和一系列角度 theta，radon 函数可以用来计算图像的 Radon 变换，在 Matlab 中也可以利用 iradon 函数来重建图像，即给出一幅图像 I 和一个角度集合 theta，使用函数 radon 计算 Radon 变换：

```
R＝radon(I,theta);
```

然后调用函数 iradon 重建图像 I：

```
IR＝iradon(R,thera)
```

在以上代码中，投影是根据原始图像 I 计算得到的，而在大多应用中，没有原始的物理图像来计算投影。例如，X 射线吸收重建，投影是通过测量放射线沿不同角度穿透物理标本的衰减过程而构造出来的。原始图像可以认为是通过切面的截面，这里图像的密度代表切片的密度。通过特殊硬件将投影图像收集起来，然后使用 iradon 函数重新构造一个内部图像，这样能够实现对生物或不透明物体进行无损照相。

Radon 逆变换通过平行波束的投影来重建图像，在平行波束的几何关系中，每个投影通过把特定角度穿过一系列线积分组合得到。图 5.20 显示了 X 射线投影重建中，平行波束的几个关系。$f(x,y)$代表图像亮度，$R_\theta(x')$代表了 θ 方向的投影。平行波束几何是应用于 X 射线吸收断层摄影中的。此时发射器和接收器数量相同，辐射的强弱代表物体的密度、质量等的积分，这相当于 Radon 变换计算中的线积分。

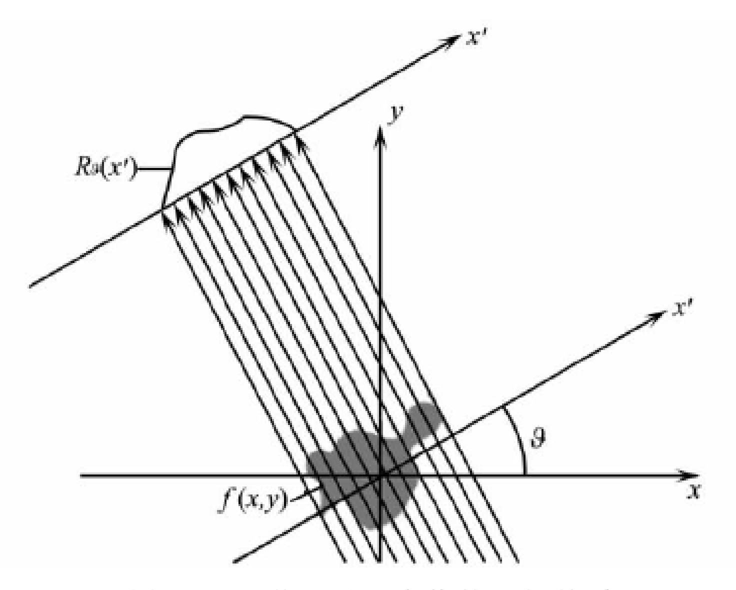


图 5.20　逆 Radon 变换的几何关系

另外一种几何关系是扇形几何关系，即扇形波束，它只有一个辐射器而有多个接收器，扇形波束投影可以转换成平行波束投影，利用 Radon 逆变换来重建图像。

iradon 函数利用过滤反向投影算法来计算 Radon 逆变换。这个算法利用 R 各列中的投影来构造图像 I 的近似值，若要获得更准确的图像，可以利用更多的投影值（theta 长度越长）。投影数越多，重新构造的图像 IR 就越接近原始图像 I。向

量 theta 必须是固定增量的，即每次角度的增加值 $\Delta\theta$ 为常数。如果知道标量 $\Delta\theta$ 以后，可作为参数取代 theta 值，传入 iradon 函数。

【例 5.13】利用 radon 函数和 iradon 函数构造一个简单图像的投影并重建图像。

说明：测试图像是 Shepp－Logan 的大脑图（见图 5.21），是由图像处理工具箱函数 phantom 创建的。Sheep－Logan 的大脑图说明了许多人脑的特性，例如，外部明亮的椭圆形外壳类似于头骨，内部的椭圆类似于大脑的内部特征或肿瘤。



图 5.21 大脑图

首先产生一个 256 灰度级的大脑图：

```
P=phantom(256);
imshow(P);
```

进行 Radon 变换的第一步是计算 3 个不同部分的大脑的 Radon 变换，$R1$ 采用 18 个投影，$R2$ 采用 36 个投影，$R3$ 采用 90 个投影。

```
theta1 = 0:10:170;theta2 = 0:5:175;theta3 = 0:2:178;
```

然后计算 3 个不同部分的大脑图的 Radon 变换。

```
[R1,xp] = radon(P,theta1);
[R2,xp] = radon(P,theta2);
[R3,xp] = radon(P,theta3);
figure, imagesc(theta3,xp,R3);
colormap(hot); colorbar
xlabel('\theta'); ylabel('x\prime');
```

最后利用不同部分的 Radon 逆变换来重构图像。

```
I1=iradon(R1,10);I2 = iradon(R2,5);I3 = iradon(R3,2);
subplot(1,3,1);
imshow(I1);title('用 R1 重构图像');
subplot(1,3,2);
imshow(I2);title('用 R2 重构图像');
subplot(1,3,3)
imshow(I3);title('用 R3 重构图像');
```

其显示结果如图 5.22 所示。

观察得到的重构图像，用 $R1$ 重构图像中有许多虚假点，重构图像效果最差，因为它使用的投影数目最少；用 $R2$ 重构图像效果较好，因为使用了 36 个投影；用 $R3$ 重构图像效果最好，最接近原图像，因为它使用了 90 个投影。所以为了使重构图像的质量提高，可以增加重构图像的投影角度的数目。

观察使用 90 个投影的 Randon 变换（见图 5.23），可以看到输入图像的一些特性。图像 Radon 变换的第一列为 0°对应图像在垂直方向的投影，中间为 90°对应

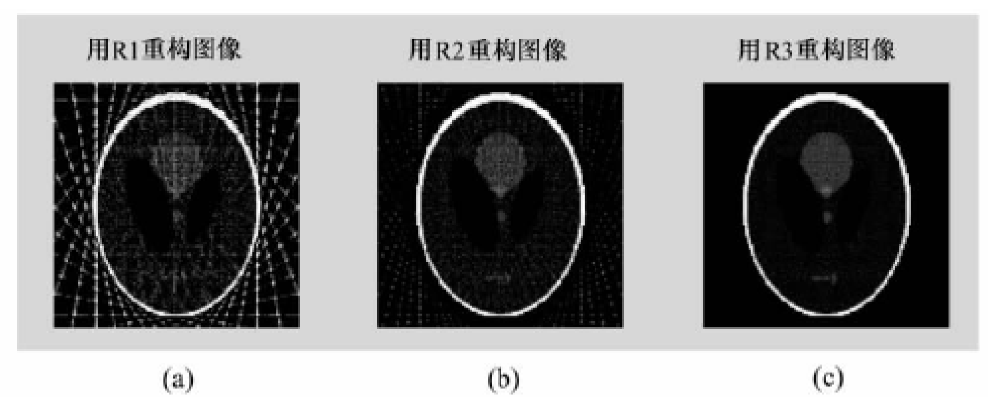


图 5.22 重构图像的比较

图像在水平方向的投影。90°的投影轮廓要宽于 0°处的投影，这是因为在大脑的椭圆处具有较大的垂直半轴。

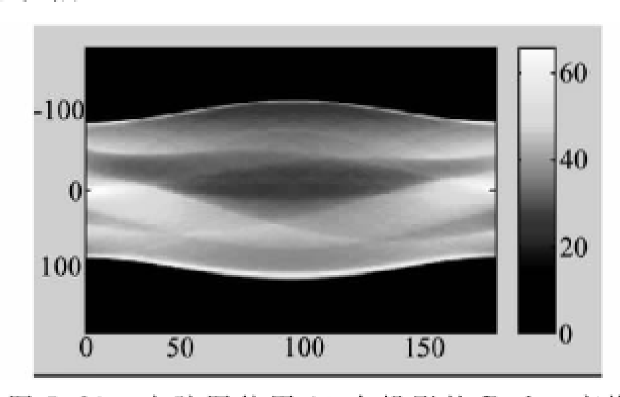


图 5.23 大脑图使用 90 个投影的 Radon 变换

5.4 小波变换

傅里叶变换能够用正弦函数之和表示任何分析函数——甚至是一个狭窄的瞬态信号。然而，这是通过错综复杂的安排，以消去一些正弦波(通过相互抵消)的方式，构造出在大部分区间都为零的函数而实现的。这对于可逆变换来说是一个有效的方法，但它却使此函数的频谱图呈现一幅相当混乱的构成。

为了克服这些缺陷，数学家和工程师们已经开发出若干种使用有限宽度基函数进行变换的方法。这些基函数不仅在频率上而且在位置上是变化的，它们是有限宽度的波，被称为小波(Wavelet)。基于小波的变换被称为小波变换(Wavelet Transforms)。正弦波和小波的比较如图 5.24 所示。

5.4.1 小波变换基础

小波是一个满足条件$\int_{-\infty}^{+\infty}\psi(t)\mathrm{d}t=0$的函数通过平移和伸缩而产生的一族函数$\psi_{a,b}(t)$，如下式：

$$\psi_{a,b}(t)=|a|^{-\frac{1}{2}}\psi(\frac{t-b}{a})\qquad a\in R,b\in R,a\neq 0 \tag{5.27}$$



图 5.24　正弦波和小波的比较
(a)正弦波;(b)小波。

按式(5.27)定义的小波称为基小波或母小波,函数族 $\psi_{a,b}(t)$ 也称为分析小波。式(5.27)中的 b 为平稳的距离,a 为伸缩的尺度。

如果任一函数 f 作周期延伸,则可以成为直线上有意义的函数,即:

$$\int_0^{2\pi} f_x(x)\mathrm{d}x < \infty \tag{5.28}$$

那么每个函数 f 均属于一个平方可积函数 $L^2(0,2\pi)$,也可以写为 $L^2(\mathbf{R})$,$\mathbf{R}$ 为实数空间。对于任一函数 $f\in L^2(\mathbf{R})$,且基本小波 $\psi\in L^2(\mathbf{R})$,那么 f 的连续小波变换的定义为:

$$W_f(a,b) = < f,\psi_{a,b} > = |\ a\ |^{\frac{1}{2}}\int_{-\infty}^{+\infty} f(t)\psi(\frac{t-b}{a})\mathrm{d}t \tag{5.29}$$

其中 $\psi(\frac{t-b}{a})$ 为基本小波的共轭函数,且 $\psi\in L^2(R)$,并满足如下条件:

$$C_\psi = \int_{-\infty}^{+\infty} \frac{|\ \hat{\psi}(\omega)\ |^2}{|\omega|}\mathrm{d}\omega < \infty \tag{5.30}$$

其中 $\hat{\psi}(\omega)$ 为 $\psi(t)$ 的傅里叶变换,该条件称为允许性条件,而 $\psi(t)$ 则称为允许小波。对任意的 $f\in L^2(\mathbf{R})$ 及 $t\in\mathbf{R}$,若 $f(t)$ 在 t 处连续,则可以由小波变换得到其反变换,得到原函数为:

$$f(t) = C_\psi^{-1}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \omega_f(a,b)\psi_{a,b}(t)\ \frac{\mathrm{d}a\mathrm{d}b}{a^2} = C_\psi < f,h > \tag{5.31}$$

设 $f\in L^2(\mathbf{R})$,$\omega_f(a,b)$ 是 f 的小波变换,则:

$$\int_{-\infty}^{+\infty} |\ f(t)\ |^2\mathrm{d}t = C_\psi^{-1}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \omega_f(a,b)\ \frac{\mathrm{d}a\mathrm{d}b}{a^2} \tag{5.32}$$

由此可以看出,小波变换是一种信息保持型的可逆变换,原来的信息保存于小波变换系数中,反变换后又可恢复。

连续小波变换有明确的物理意义,b 的作用是确定对 $f(t)$ 分析的时间位置,即时间中心。尺度因子 a 的作用是把基本小波 $\psi(t)$ 作伸缩。当 $\psi(t)$ 变成 $\psi(\frac{t}{a})$,当 $a>1$ 时,若 a 越大,则 $\psi(\frac{t}{a})$ 的时域支撑范围(即时域宽度)较之 $\psi(t)$ 变得越大;反之,当 $a<1$ 时,a 越小,则 $\psi(\frac{t}{a})$ 的宽度越窄。这样,a 和 b 联合起来确定了对 $f(t)$

分析的中心位置及分析的时间宽度，如图 5.25 所示。

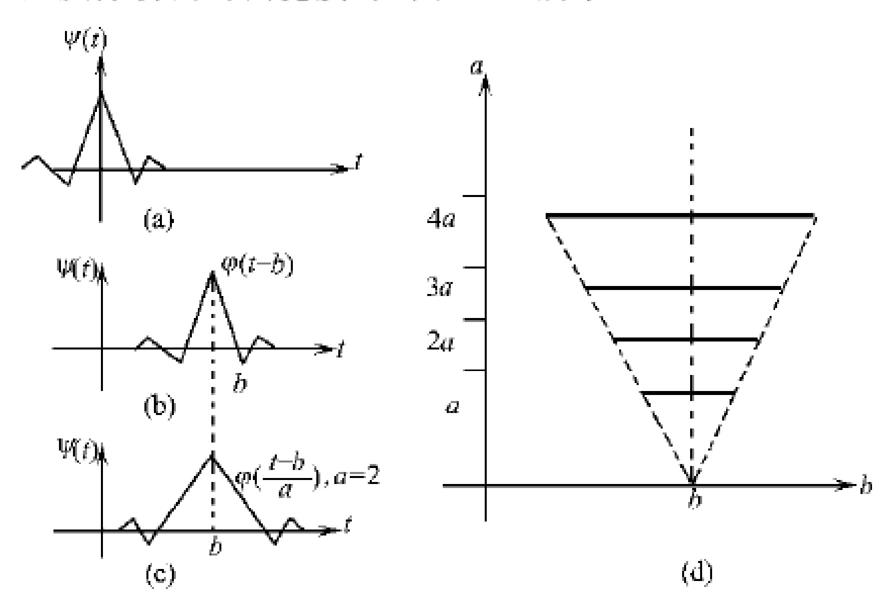


图 5.25　基本小波的伸缩及参数 a 和 b 对分析范围的控制

(a) 基本小波；(b) $b>0$，$a=1$；(c) b 不变，$a=2$；(d) 分析范围。

小波变换与傅里叶变换的思想基本一致，就是信号在由一族基函数张成的空间的投影中来表征信号。但是，这一族函数具有一个显著的特点，即该函数系是通过一个基本小波函数的不同尺度的伸缩和平移构成的，其时宽与带宽的乘积很小，而且在时间和空间上很集中。

在各个学科应用领域中若要采用小波变换，就有一个变换基的选择问题，这点与经典的傅里叶变换不一样，傅里叶变换基是固定的。在具体应用中需要根据原函数 $f(t)$的特点来选择小波变换基 $\psi(t)$，使小波变换能更好地反映 $f(t)$的特征。对小波理论发展有重大贡献的科学家们构造了正交小波基库，以供选择。除此之外，他们又构造了非正交小波基。

5.4.2　离散小波变换

在图像处理中，需把连续小波及其小波变换离散化才有意义。作为一种方便的形式，计算机实现中往往对小波进行二进制离散，并把经这种离散化后的小波及相应的小波变换称为离散小波变换(DWT)离散小波变换是对连续小波变换的尺度和位移按照 2 的幂次进行离散化得到的，故又称为二进制小波变换。

在每个可能的尺度下计算小波系数，计算量相当大，将产生惊人的数据，于是考虑选择部分膨胀和位移来进行计算。使用二进制膨胀和位移，就使分析十分有效，并且相当精确。

虽然傅里叶变换可以反映信号的整个内涵，但是表现形式却不够直观，而且噪声会导致信号频谱复杂化。在信号处理领域一直都是使用一族带通滤波器将信号分解为不同频率分量，即将输入信号 $f(x)$并行送到带通滤波器族 $H_i(x)$中。假设

每一个滤波器相应的输入为 $g_i(x)$，则：

$$g_i(x) = \int_{-\infty}^{+\infty} f(t) H_i(x-t) \mathrm{d}t \tag{5.33}$$

构造一族滤波器 $H_i(x)$ 时要满足如下条件：

$$\sum_{i=1}^{\infty} H_i(x) = 1 \Rightarrow \sum_{i=1}^{\infty} g_i(x) = f(x) \tag{5.34}$$

这里 $g_i(x)$ 就是一组小波变换系数，而 $\{H_i(x)\}$ 就是一个小波函数集。

滤波器族理论提供了一个振荡信号分析的方便手段，但是在图像分析中，人们关心的并不是真正的振荡信号，而是信号的一个或一部分周期。为了更仔细地观察这部分信息，可以使用不同的小波变换尺度来膨胀或收缩小波基，从而实现信号的多分辨率。

小波变换可以用在图像处理中的很多地方，可以进行基于小波变换的图像分解与重构。

1. 近似与细节

实际上，人们是在一定尺度上认识信号的，人的感觉器官和物理仪器都有一定的分辨力。对于低于一定尺度的信号的细节是无法认识的，因此对低于一定尺度信号的研究也是没有意义的。小波分解的意义就在于能够在不同尺度上对信号进行分解，而且对不同尺度的选择可以根据不同的目的来确定。

对于许多信号，低频成分相当重要，它常蕴含着信号的特征，而高频成分则给出信号的细节或差别。例如，人的话音如果除去高频成分，听起来则有所不同，但仍能知道所说的内容，然而，除去足够的低频成分，则听到的是一些无意义的声音。在小波分析中常用到近似与细节。近似表示信号的高尺度，即低频成分；而细节表示的是低尺度，即高频成分。因而原始信号通过两个相互滤波器产生两个信号。

一个信号若采用图 5.26 所示的方法，理论上将产生两倍于原始数据的信号量，为此采用以下采样的方法来减少数据量，通过计算 DWT 系数可得到原始信号的近似与细节。

2. 信号多层分解

通过不断的分解过程，将近似信号连续分解，就可将信号分解成许多低分辨力成分。图 5.27 就是这样一个小波分解树。图中 S 表示原始信号，A 表示近似信号，B 表示细节信号，下标表示分级的层数，$S = A_1 + B_1 = A_2 + B_2 + B_1 = A_3 + B_3 + B_2 + B_1$。

由于分解过程是迭代的，从理论上讲可以无限制地连续分解下去。但事实上，分解可以进行到细节只包含单个样本为止。因此在实际应用中，一般依据信号的特征或者合适的标准来选择适当的分解层数。

【例 5.14】对电源信号进行小波分解和重构。

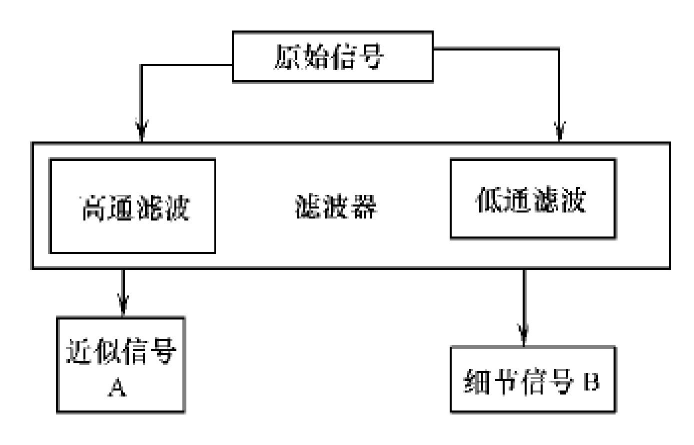


图 5.26　信号一阶滤波示意图

说明：本例中，读入信号，并对信号进行一层小波分解得到高尺度低频分量 cA1 和低尺度高频分量 cD1，显示结果如图 5.28 所示。

图 5.27　小波分解树

```
load leleccum;
s = leleccum(1:3920);
l_s = length(s);
subplot(1,3,1);plot(s);title('原始电源信号');
[cA1,cD1] = dwt(s,'db1');          %进行小波分解
subplot(1,3,2);plot(cA1);title('一层分解的近似分量');
subplot(1,3,3);plot(cD1);title('一层分解的细节分量');
```



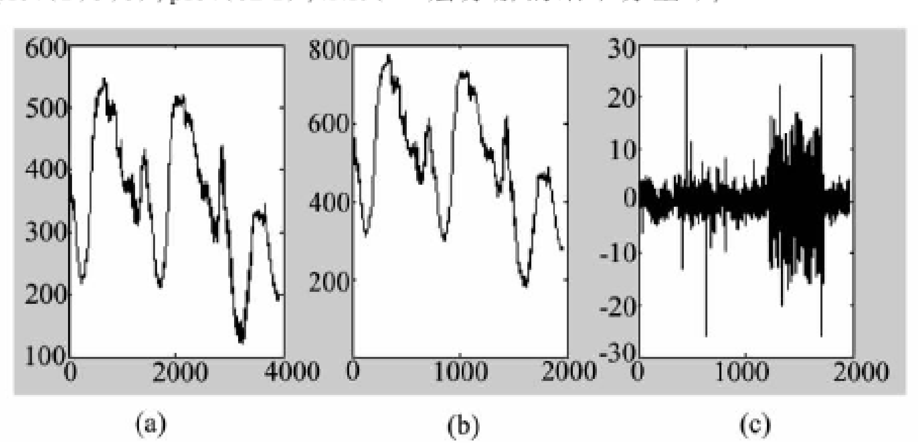


图 5.28　原始信号和一层小波变换结果

(a)原始电源信号；(b)高尺度低频分量；(c)低尺度高频分量。

将得到的一层低频分量和高频分量重构为细节信号和近似信号，如图 5.29 所示。

```
A1 = upcoef('a',cA1,'db1',1,l_s);          %从系数得到近似信号
D1 = upcoef('d',cD1,'db1',1,l_s);          %从系数得到细节信号
```

将重构的细节信号和近似信号相加，得到重构的图像，并和原始图像进行比

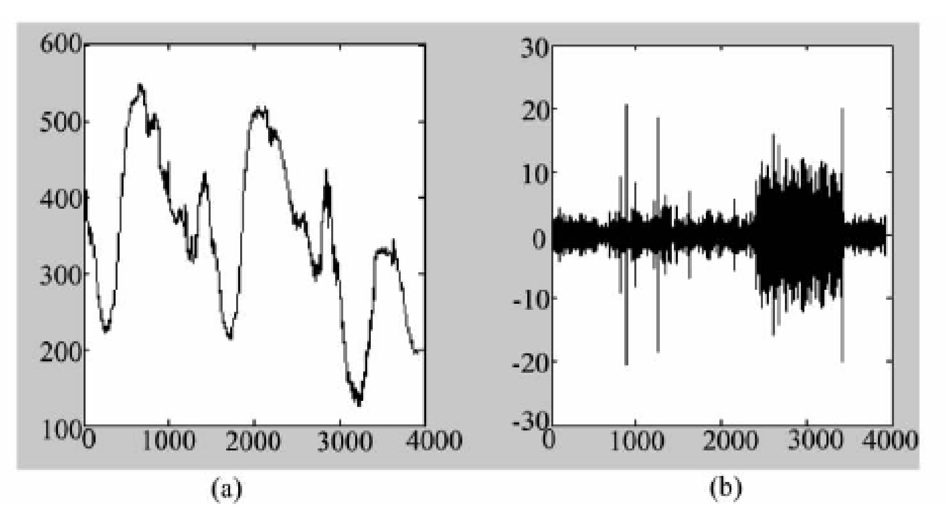


图 5.29　重构的近似信号和细节信号

(a)近似信号；(b)细节信号。

较，如图 5.30 所示。

```
subplot(1,2,1);plot(s);
subplot(1,2,2);plot(A1+D1);
```

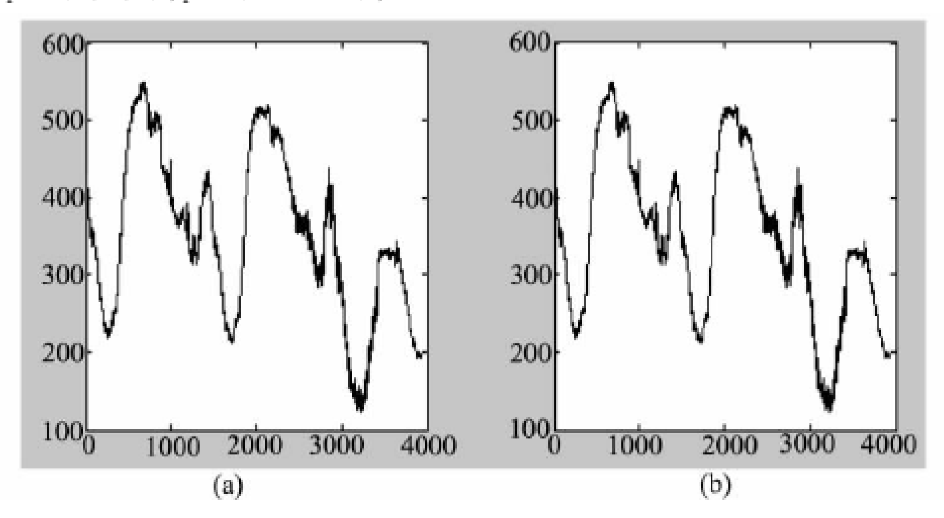


图 5.30　原始信号和重构信号的比较

(a)原始信号；(b)重构信号。

通过比较可以看出小波重构信号和原始信号没有差异。

小波分解可以使人们在任意尺度观察信号，只要使用小波函数的尺度合适。小波分解将信号分解为近似分量和细节分量，它们在应用中分别有不同的特点。比如，对于含噪信号，噪声分量的主要能量一般集中在小波分解的细节分量中，因此对细节分量进行阈值处理可以过滤噪声。

5.4.3　小波分析在图像处理中的应用

图像处理是小波分析应用的重要领域，近年来小波分析已经被证明是进行图像处理强有力的工具之一。因为，小波分解可以把图像分层次按照小波基展开，并且可以根据图像的性质及给定的图像处理标准确定展开到哪一级为止，还可以把细节分量和近似分量分开，所以小波分析可以用于压缩、去噪等方面。

表 5.2～表 5.5 列出了小波工具箱中实现小波变换的一些函数。

表 5.2　二维小波分解函数

函数名	功　能
Dwt2	单层二维小波分解
Dwtper2	单层二维离散小波变换(周期性)
Wavedec2	多层二维小波分解

表 5.4　二维小波分解结构应用函数

函数名	功　能
detcoef2	提取二维小波分解高频系数
appcoef2	提取二维小波分解低频系数

表 5.3　二维小波重建函数

函数名	功　能
idwt	单层二维小波重构
idwtper2	单层二维离散逆小波分析(周期性)
waverec2	多层二维小波重构
upwlev2	二维小波分解的单层重构
wrcoef2	二维小被分解系数单层重构
upcoef2	二维小波分解的直接重构

表 5.5　二维小波消噪和压缩函数

函数名	功　能
Wthresh	进行软阈值或硬阈值处理
wthcoef2	二维信号的小波系数阈值处理
Ddencmp	获取在消噪或压缩过程中的默认阈值(软或硬)、熵标准
wdencmp	用小波进行信号的消噪和压缩(也可以用于一维的情况)

1．小波分解

(1) 函数 dwt2。

功能:进行一层二维离散小波变换。

语法格式:

```
[cA,cH,cV,cD] = dwt2(X,'wname')
[cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)
```

说明:

X 表示输入信号;wname 表示使用的小波基函数;cA 是分解得到的近似分量系数;cH 是分解得到的水平分量系数;cV 是分解得到的垂直分量系数;cD 是分解得到的对角线分量系数;Lo_D 是指定的低通滤波器组;Hi_D 是指定的高通滤波器组。

图 5.31 示意可在第 j 层分解信号,对第 j 层的近似分量 cA_j 进行小波变换,

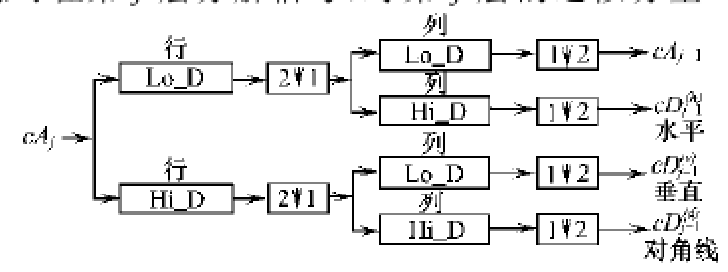


图 5.31　分解信号示意图

得到了 $j+1$ 层的分解信号。

（2）函数 idwt2。

功能：进行一层二维小波反变换。

语法格式：

```
X=idwt2(cA,cH,cV,cD,'wname')
X=idwt2(cA,cH,cV,cD,Lo_R,Hi_R)
X=idwt2(cA,cH,cV,cD,'wname',S)
X=idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)
```

说明：

X 是重构的分量信号；wname 表示使用的小波基函数；cA 是分解得到的近似分量系数，cH 是分解得到的水平分量系数；cV 是分解得到的垂直分量系数；cD 是分解得到的对角线分量系数；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组。

（3）函数 upcoef2。

功能：重构二维小波分解的各种分量。

语法格式：

```
Y=upcoef(O,X,'wname',N)
Y=upcoef(O,X,'wname',N,L)
Y=upcoef(O,X,Lo_R,Hi_R,N)
Y=upcoef(O,X,Lo_R,Hi_R,N,L)
Y=upcoef(O,X,'wname')
Y=upcoef(O,X,Lo_R,Hi_R)
```

说明：

Y 是重构的细节信号分量；O 是细节信号的类型，为"a"表示近似细节信号，为"h"表示水平细节信号，为"v"表示垂直细节信号，为"d"表示对角线细节信号；X 是细节信号；wname 是使用的小波基函数；N 表示分解的层数，默认值为 1；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组。

（4）函数 wavedec2。

功能：进行多层二维小波分解。

语法格式：

```
[C,S]=wavedec2(X,N,'wname')
[C,S]=wavedec2(X,N,Lo_R,Hi_R)
```

说明：

X 是输入信号；N 表示分解的层数，默认值为 1；wname 是使用的小波基函数；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组；C 和 S 是分解得到的向量和对应矩阵。

（5）函数 waverec2。

功能：进行多层二维小波分解的重构。

语法格式：

```
X=waverec2(C,S,'wname')
X=waverec2(C,S,Lo_R,Hi_R)
```

说明：

X 是重构的原始信号；C 和 S 是分解得到的向量和对应矩阵；wname 是使用的小波基函数；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组。

（6）函数 wrcoef2。

功能：用分解得到的 C、S 进行多层二维小波分解某一层的重构。

语法格式：

```
X=wrcoef2('type',C,S,'wname',N)
X=wrcoef2('type',C,S,Lo_R,Hi_R,N)
X=wrcoef2('type',C,S,'wname')
X=wrcoef2('type',C,S,Lo_R,Hi_R)
```

说明：

X 是重构的分量信号；type 是分量类型，为“a”表示近似分量，为“h”表示水平分量，为“v”表示垂直分量，为“d”表示细节分量；N 表示重构的层次，默认值是 size(S,1)－2；wname 是使用的小波基函数；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组。

（7）函数 appcoef2。

功能：提取多层二维小波分解的近似分量。

语法格式：

```
A=appcoef(C,S,'wname',N)
A=appcoef(C,S,'wname')
A=appcoef(C,S,Lo_R,Hi_R)
A=appcoef(C,S,Lo_R,Hi_R,N)
```

说明：

A 是得到的近似分量；C 和 S 是函数 wavedec2 得到的分解结构；wname 是使用的小波基函数；N 是分解的层数；Lo_R 是指定的低通滤波器组；Hi_R 是指定的高通滤波器组。

（8）函数 detcoef2。

功能：提取多层二维小波分解的细节分量。

语法格式：

```
D=detcoef2(O,C,S,N)
```

说明：

D 是得到的分量;O 是细节信号的类型,为“h”表示水平细节信号,为“v”表示垂直细节信号,为“d”表示对角线细节信号;N 表示分解的层数;C 和 S 是函数 wavedec2 分解得到的结果。

【例 5.15】用小波变换对图像进行一维、二维分解和压缩。

说明:在这个例子中将介绍以上函数的使用。

第一步 如图 5.32 所示,读入图像并显示。

```
load wbarb;
image(X); colormap(map); colorbar;
```

第二步 对图像进行第一层次的小波分解。

```
[cA1,cH1,cV1,cD1] = dwt2(X,'bior3.7');
%进行一层小波分解
```

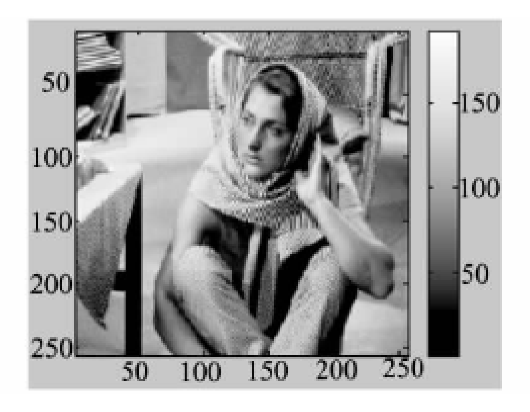


图 5.32 原始图像

第三步 对小波图像第一层次的分解信息进行重构并显示,结果如图 5.33 所示。

```
A1=upcoef2('a',cA1,'bior3.7',1); %重构细节分量信号
H1=upcoef2('h',cH1,'bior3.7',1); %重构水平分量信号
V1=upcoef2('v',cV1,'bior3.7',1); %重构垂直分量信号
D1=upcoef2('d',cD1,'bior3.7',1); %重构对角线分量信号
colormap(map);
subplot(2,2,1);image(wcodemat(A1,192));title('细节分量');
subplot(2,2,2);image(wcodemat(H1,192));title('水平分量');
subplot(2,2,3);image(wcodemat(V1,192));title('垂直分量');
subplot(2,2,4);image(wcodemat(D1,192));title('对角线分量');
```

第四步 用一层分解信息重构。

```
Xsyn = idwt2(cA1,cH1,cV1,cD1,'bior3.7');
```

第五步 对原始图像进行二层小波分解。

```
[C,S] = wavedec2(X,2,'bior3.7');        %得到分解结构[c,s]
cA2 = appcoef2(C,S,'bior3.7',2);        %得到细节分量系数
cH2 = detcoef2('h',C,S,2);              %得到二层水平分量系数
cV2 = detcoef2('v',C,S,2);              %得到二层垂直分量系数
cD2 = detcoef2('d',C,S,2);              %得到二层对角线分量系数
cH1 = detcoef2('h',C,S,1);              %得到一层水平分量系数
cV1 = detcoef2('v',C,S,1);              %得到一层垂直分量系数
cD1 = detcoef2('d',C,S,1);              %得到一层对角线分量系数
```

第六步 重构二层分解的各个分量信号。

```
A2 = wrcoef2('a',C,S,'bior3.7',2);      %重构二层细节分量
H1 = wrcoef2('h',C,S,'bior3.7',1);      %重构一层水平分量
```

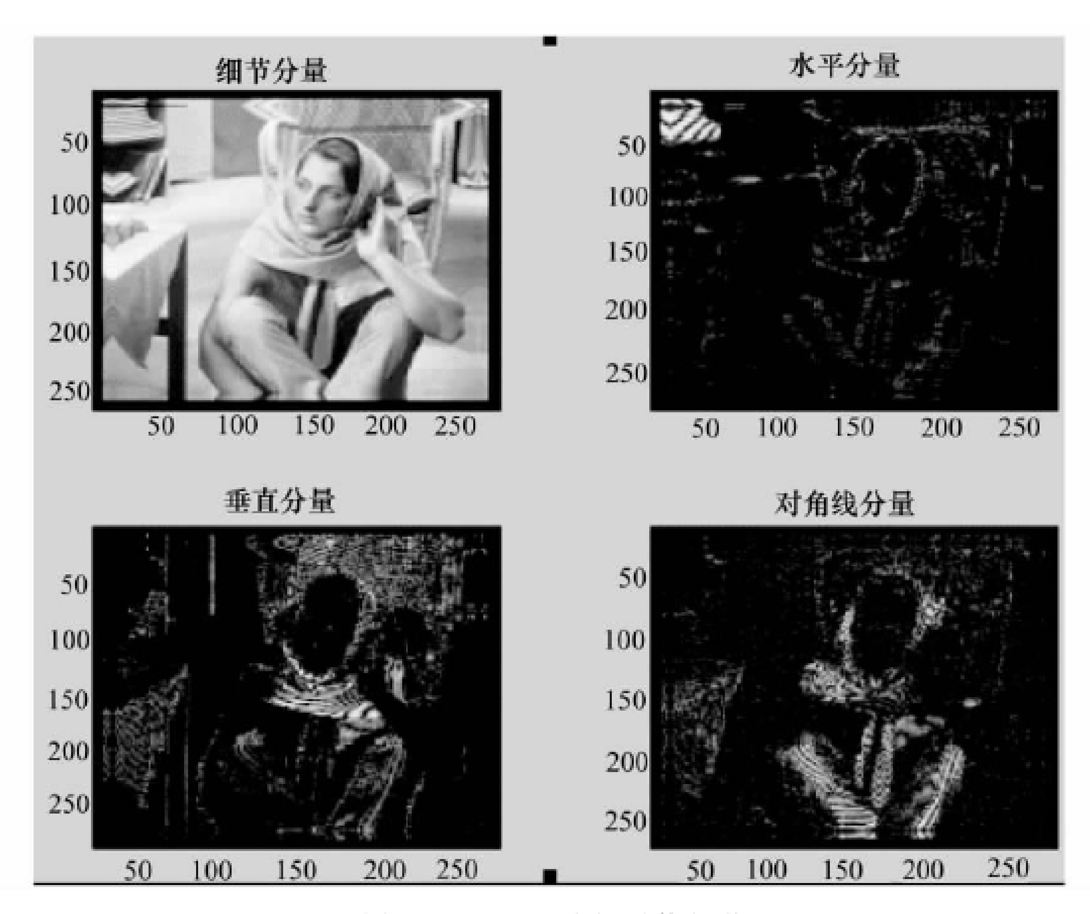


图 5.33　一层分解重构细节

```
V1 = wrcoef2('v',C,S,'bior3.7',1);      %重构一层垂直分量
D1 = wrcoef2('d',C,S,'bior3.7',1);      %重构一层对角线分量
H2 = wrcoef2('h',C,S,'bior3.7',2);      %重构二层水平分量
V2 = wrcoef2('v',C,S,'bior3.7',2);      %重构二层垂直分量
D2 = wrcoef2('d',C,S,'bior3.7',2);      %重构二层对角线分量
```

第七步 显示二维分解的图像，如图 5.34 所示。

```
colormap(map);
subplot(2,4,1);image(wcodemat(A1,192));
title('一层近似分量')
subplot(2,4,2);image(wcodemat(H1,192));
title('一层水平分量')
subplot(2,4,3);image(wcodemat(V1,192));
title('一层垂直分量')
subplot(2,4,4);image(wcodemat(D1,192));
title('一层对角线分量')
subplot(2,4,5);image(wcodemat(A2,192));
title('二层近似分量')
subplot(2,4,6);image(wcodemat(H2,192));
title('二层水平分量')
subplot(2,4,7);image(wcodemat(V2,192));
```

```
title('二层垂直分量')
subplot(2,4,8);image(wcodemat(D2,192));
title('二层对角线分量')
```

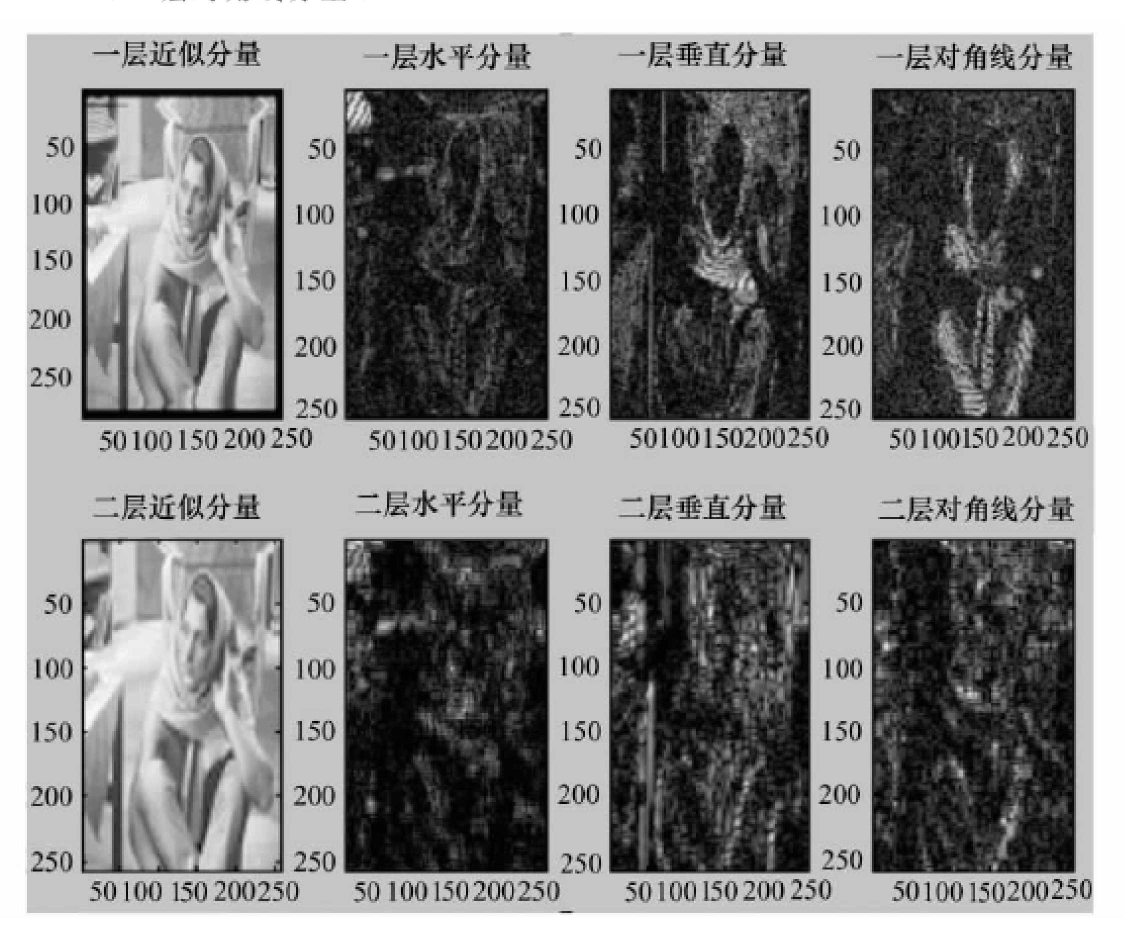


图 5.34 二层分解重构细节

第八步 重构原始图像。

```
X0 = waverec2(C,S,'bior3.7');
```

2. 小波压缩

在小波工具箱中，提供了 ddencmp 和 wdencmp 函数来对图像进行压缩和去噪声。

(1) 函数 ddencmp。

功能：求去噪或压缩的默认值。

语法格式：

```
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)
```

说明：

THR 表示阈值；SORTH 表示软、硬阈值；KEEPAPP 表示允许保留近似系数；CRIT 表示熵名；IN1 表示压缩或者去噪，为“den”表示去噪，为“cmp”表示压缩；IN2 为“wv”表示使用小波，为“wp”表示小波包；X 为输入图像。

(2) 函数 wdencmp。

功能：对输入图像进行压缩或者去噪。

语法格式：

[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('gbl',X,'wname',N,THR,SORH,KEEPAPP)

[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH)

[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',C,L,'wname',N,THR,SORH)

说明：

XC是返回的压缩或者去噪信号；[CXC,LXC]表示XC的小波分解结构；PERF0和PERFL2表示恢复和压缩率；gbl表示使用全局正阈值THR；lvd表示使用包含在THR中的层相关阈值；[C,L]表示小波分解结构；KEEPAPP=1表示对近似系数也阈值化。

【例5.16】用小波变换对图像tire进行压缩。

```
load tire;
subplot(2,2,1);image(X);
colormap(map); title('原始图像');
axis square;
%使用db3小波基对图像进行二层小波分解
[c,s]=wavedec2(X,2,'db3');
%使用ddencmp函数得到压缩的默认值
[thr,sorh,keepapp]=ddencmp('cmp','wv',X);
%将ddencmp函数得到的阈值gbl作为全局阈值,用来对所有高频系数进行相同
%的阈值量化处理
[Xcomp,cxc,lxc,perf0,perfl2]=wdencmp('gbl',c,s,'db3',2,thr,sorh,keepapp);
%将压缩后的图像与原始图像相比较,并显示出来
subplot(2,2,2);image(Xcomp);colormap(map);
title('压缩后图像');
axis square;
disp('小波分解系数中置0的系数个数百分比:');
perf0
disp('压缩后图像剩余能量百分比:')
perfl2
```

其显示结果如图5.35所示。

工作台窗口中显示：

```
小波分解系数中置0的系数个数百分比:perf0 = 48.7276
压缩后图像剩余能量百分比:          perfl2 = 99.9930
```

可以看到压缩后的图像保留了原始图像49%的系数，但是却保留了99.99%以上的能量，得到了很好的压缩效果。

3. 图像融合

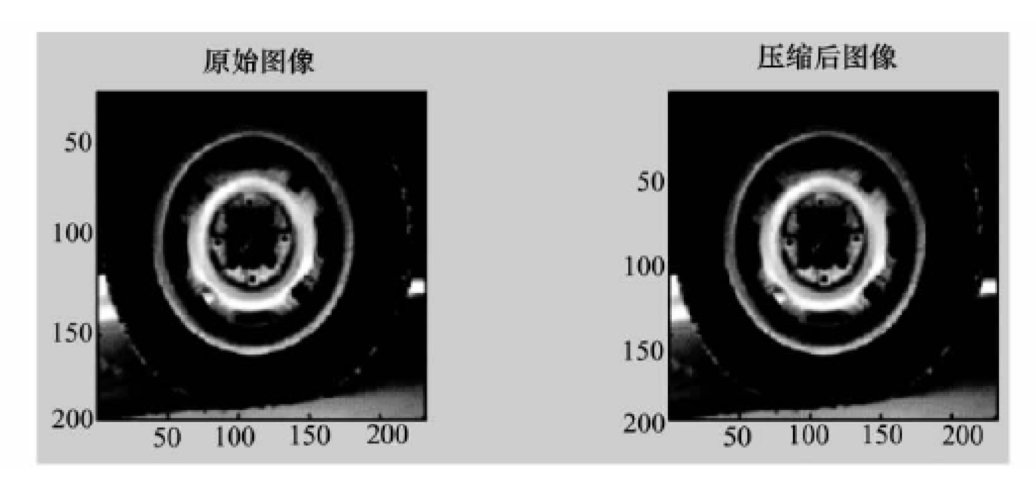


图 5.35　小波压缩图像

图像融合是将同一对象的两个或更多的图像合成在一幅图像中，以使它比原来的任何一幅更容易为人们所理解。这一技术可应用于多频谱图像理解以及医学图像处理等领域，在这些场合，同一物体部件的图像往往是采用不同的成像机理得到的。

【例 5.17】用二维小波分析将图像 woman.mat 和 wbarb.mat 融合在一起。

说明：在融合的过程中，可以队图像进行增强、去噪或对比度变换等处理。运用小波进行图像融合一般按照以下步骤进行：

(1) 首先运用小波分解将图像分解到小波域；

(2) 然后在小波变换域里将分解的系数进行融合；

(3) 最后把频域图像变换回空域。

```
load woman;
X1=X;map1=map;
subplot(2,2,1);image(X1);colormap(map1);
title('图像 woman');
load wbarb;
X2=X;map2=map;
subplot(2,2,2);image(X2);colormap(map2);
title('图像 wbarb');
%使用'sym4'小波基对图像 X1 进行二层小波分解
[c1,s1]=wavedec2(X1,2,'sym4');
sizecl=size(c1);
%对分解系数进行处理,通过处理,突出轮廓部分,弱化细节部分
for i=1:sizecl(2)
        c1(i)=1.2*c1(i);
end
%使用'sym4'小波基对图像 X2 进行二层小波分解
[c2,s2]=wavedec2(X2,2,'sym4');
```

```
%在小波变换域中对图像系数进行融合
c=c1+c2;
%对融合的系数进行重构,得到融合后的图像
xx=waverec2(c,s1,'sym4');
%显示融合的图像
subplot(2,2,3);image(xx);
title('融合图像 1');
%由于融合图像亮度过高,所以做一些处理,减小图像亮度
c=0.5*c;
%对融合的系数进行重构
xx=waverec2(c,s1,'sym4');
%显示重构后的图像
subplot(2,2,4);image(xx);
title('融合图像 2');
```

其显示结果如图 5.36 所示。

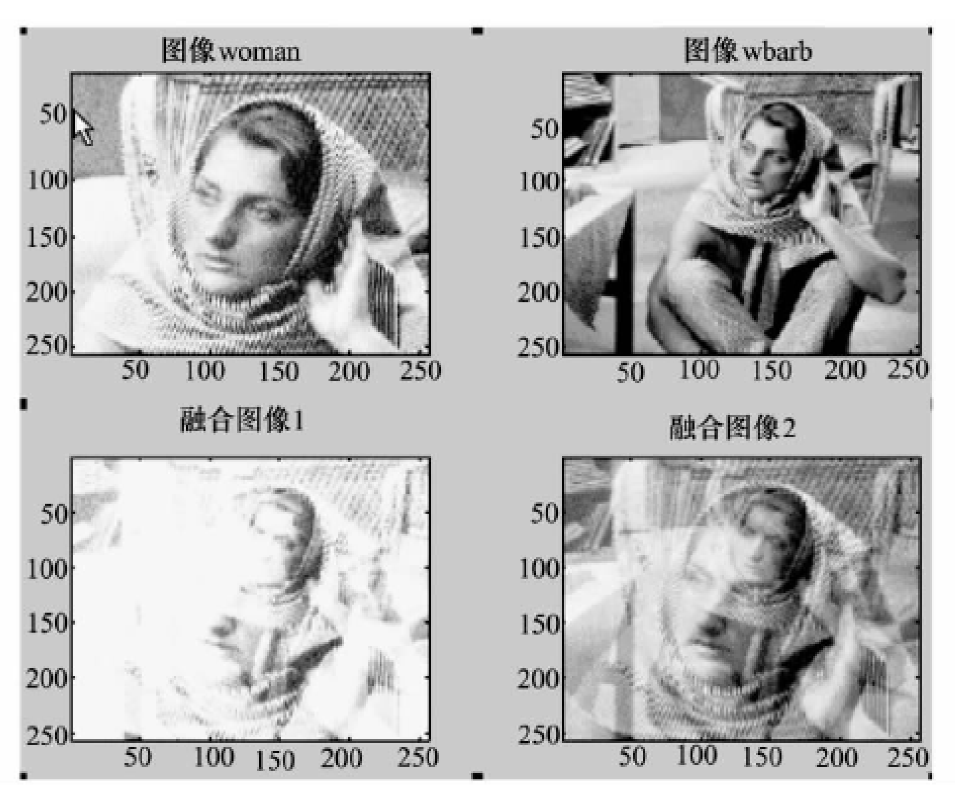


图 5.36　小波图像融合

4. 图像去噪

【例 5.18】给定一个二维信号(文件名为 sinsin.mat),利用小波分析对信号进行去噪处理。

说明:该问题是一个用二维小波分析进行信号去噪处理的典型实例,这里用 Matlab 所提供的 ddencmp 和 wdencmp 去噪函数实现去噪,其去噪过程可以按照如下程序进行。

```
%装入图像
load sinsin
%显示原始图像
subplot(1,3,1);image(X);
title('原始图像');
%对图像加入噪声
init= 2055615866; randn('seed',init);
x = X + 15 * randn(size(X));
%显示含噪声图像
subplot(1,3,2);image(x);
title('含噪声图像');
%下面进行图像的去噪处理,在这里使用的是软阈值,近似细节分量被保留下来
%使用 ddencmp 函数来计算去噪的默认阈值和熵标准
[thr,sorh,keepapp] = ddencmp('den','wv',x);
%使用 wdencmp 函数来实现图像去噪,在这里使用了全局阈值
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
%显示去噪后的图像
subplot(1,3,3);image(xd);
title('去除噪后的图像');
```

其显示结果如图 5.37 所示。

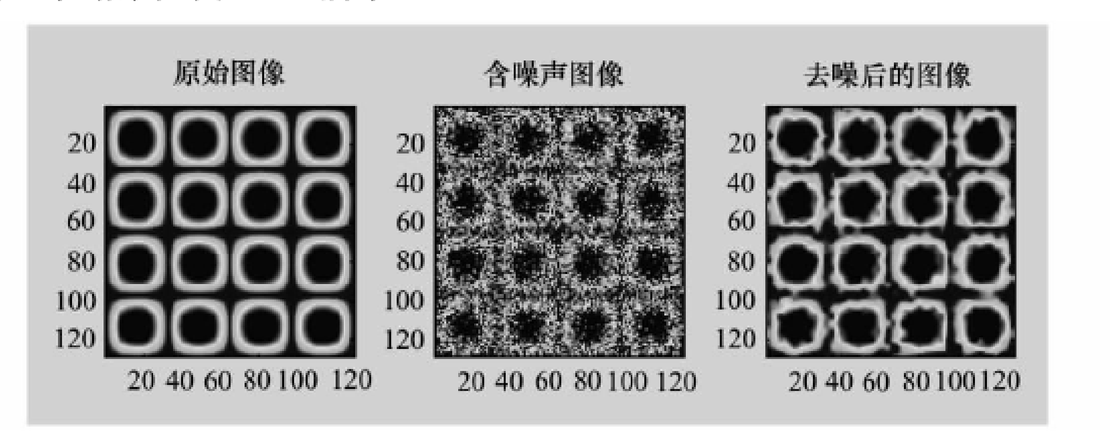


图 5.37　小波图像去噪

从上面三个图像的比较可以看出,Matlab 中的 ddencmp 和 wdencmp 函数可以有效地进行去噪处理。

【例 6.19】对含噪图像用二维小波分析和图像的中值滤波进行图像的平滑。

分析:这个问题是一个图像平滑处理问题。首先,在频域内对图像进行增强,然后在空间域内加入较大的白噪声。通过对含噪图像进行平滑处理,可以使含噪图像具有较好的平滑效果。其具体处理过程如以下程序:

```
%读取原始图像
load woman;
```

```
%用 sym4 小波基对图像进行小波二层分解
[c,s]=wavedec2(X,2,'sym4');
sizec=size(c);
%通过高频率,低频减弱,在频域里对图像进行增强
for i=1:sizec(2)
    if(c(i)>350)
        c(i)=1.2*c(i)
    else
        c(i)=0.5*c(i)
        end
end
%对经过增强处理后的系数进行重构
xx=waverec2(c,s,'sym4');
%加入噪声
init=2788605826
rand('seed',init);xx=xx+68*(rand(size(xx)));
subplot(2,2,1);image(xx);
title('增强的含噪图像');
axis square;

%采用 3×3 的模板图像进行中值滤波处理
for i=2:1:255
        for j=2:1:255
            a=0;
            for m=1:3
                for n=1:3
                    a=a+xx(i+m-2,j+n-2)
                end
            end
            a=a/9;
            xx(i,j)=a;
        end
end
%画出平滑后的图像
colormap(map);
subplot(2,2,2);image(xx);
axis square;
title('平滑后的图像');
```

其显示结果如图 5.38 所示。

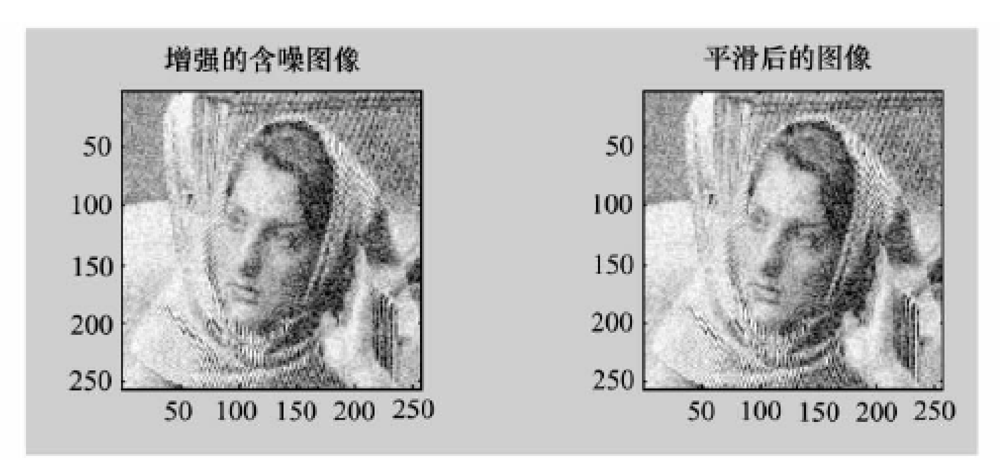


图 5.38 小波平滑图像

5.5 图像的变换在图像压缩中的应用

5.5.1 图像压缩概述

数据压缩最初是信息论研究中的一个重要课题，在信息论中被称为信源编码。现在，数据压缩已不限于编码方法的探讨和研究，它已逐步形成一个相对独立的体系，并且还扩展到研究数据的表示、传输及转换方法，目的是不断减少数据存储所占的空间和数据传输所需要的时间，从而达到提高工作效率、降低系统工作成本的目的。

对于图像来说，如果需要进行快速或实时传输以及大量存储，就需要对图像数据进行压缩。在同等的通信容量下，如果图像数据压缩后再传输，就可以传输更多的图像信息，也就可以增加通信的能力，例如，用普通的电话线传输图像信息。图像压缩研究的就是寻找高压缩比的方法且压缩后的图像要有合适的信噪比，在压缩传输后还要恢复原信号。并且在压缩、传输、恢复的过程中，还要求图像的失真度小。

现在的图像压缩技术一般基本基于如下两点：

(1) 图像信息存在着很大的冗余度，数据之间存在着相关性，如相邻像素之间色彩基本相同；

(2) 人的视觉对于边缘急剧变化不敏感，人眼具有对图像的亮度敏感高、对色度敏感性低的特点。

图像压缩主要就是根据这两点特征进行的，由此发展出图像压缩的两类基本方法：一种是将相同的或相似的数据或数据特征归类，使用较少的数据量描述原始数据，达到减少数据量的目的，这种压缩一般为无损压缩；另一种是利用人眼的视觉特性有针对性地简化不重要的数据，以减少总的数据量，这种压缩一般为有损压

缩,只要损失的数据不太影响人眼主观接收的效果,就可采用。

图像压缩的主要参数之一是图像压缩比,定义为压缩前的图像数据量与压缩后的图像数据量之比,即:

$$\text{图像数据压缩比} = \frac{\text{压缩后的图像数据量}}{\text{压缩前的图像数据量}} \tag{5.35}$$

显然,压缩比越小,压缩后的图像文件数据量越小,图像质量有可能损失越多。但压缩比并不是一个绝对的指标,压缩的效果还与压缩前的图像效果及压缩方法有关。

图像数据总是存在各种信息的冗余,如空间冗余、信息冗余、视觉冗余和结构冗余等。压缩就是去掉各种冗余,保留有用的信息。图像压缩的过程常称为编码,图像恢复的过程常称为解码。

现今,图像压缩方面的标准主要有 H.26X 系列(H.261 和 H.263)、JPEG、MPEG 系列(MPEG－1,MPEG－2,MPEG－4 和计划中的 MPEG－7)。

1. H.261 标准

H.261 标准简称为 p64 标准,由国际电报电话咨询委员会(CCITT)的一个专家组在 1990 年 12 月制定的。其核心思想是利用离散余弦变换及运动补偿算法,通过减少每帧图像间时间上和空间上的冗余性和相关性信息来减少数据量。H.261标准适合在 64Kb/s～384Kb/s 的低带宽下传输实时视频图像,但图像质量不理想。

2. JPEG 标准

JPEG 全名为 Joint Photographic Experts Group(联合摄影专家组),它是一个在国际标准组织(ISO)下从事静态图像压缩标准制定的委员会。JPEG 标准完成于 1992 年,相当于 ISO/IEC 国际标准 10928－1 以及 ITU－T(国际电信同盟)T.81 标准。JPEG 标准基于离散余弦变换,适用于一般连续色调、多级灰度、彩色或黑白静止图像压缩,有多种编码格式和数据格式,可以用于彩色传真、静止图像、可视通信、印刷出版、新闻图片、医学和卫星图像的传输。

3. MPEG－1 标准

MPEG 是 Moving Picture Expert Group(运动图像专家组)的缩写,即 ISO11172。

MPEG－1 标准制定于 1992 年,可适用于不同带宽的设备,如 CD－ROM(光盘驱动器)、Video－CD(激光视频光盘)、CD－i(交互式 CD)。它的目的是把 221Mb/s 的 NTSC(全国系统电视委员会)图像压缩到 1.2Mb/s,压缩率为 200∶1。这是图像压缩的工业认可标准。它可针对 SIF(中音频)标准分辨率(对于 NTSC 制为 352×240;对于 PAL(逐行倒相)制为 352×288)的图像进行压缩,传输速率为 1.5Mb/s,每秒播放 30 帧,具有 CD 音质,质量级别基本与 VHS(广播级录像

带)相当。MPEG 的编码速率最高可达 4Mb/s～5Mb/s,但随着速率的提高,其解码后的图像质量有所降低。

应用 MPEG－1 技术最成功的产品非 VCD(视频高密光盘) 莫属,VCD 作为价格低廉的影像播放设备,得到广泛的应用和普及。MPEG－1 标准也被用于数字电话网络上的视频传输,如 ADSL(非对称数字用户线路),VOD(视频点播)以及教育网络等。

4. MPEG－2 标准

MPEG－2 标准制定于 1994 年,设计目标是高级工业标准的图像质量以及更高的传输率。MPEG－2 标准所能提供的传输率为 3Mb/s～10Mb/s,在 NTSC 制式下的分辨率可达 720×486,MPEG－2 标准能够提供广播级的视像和 CD 级的音质。MPEG－2 标准的音频编码可提供左右中及两个环绕声道以及一个加重低音声道和多达七个伴音声道。MPEG－2 标准的另一特点是,可提供一个较广范围的可变压缩比,以适应不同的画面质量、存储容量以及带宽的要求。

MPEG－2 标准技术就是实现 DVD(数字化视频光盘)的标准技术,现在 DVD 播放器也开始在家庭中普及起来了。除了作为 DVD 的指定标准外,MPEG－2 标准还可用于为广播、有线电视网、电缆网络以及卫星直播提供广播级的数字视频。

5. MPEG－3 标准

由于 MPEG－2 标准的出色性能表现已能适用于 HDTV(高清晰度电视),使得原打算为 HDTV 设计的 MPEG－3 标准还没产生就被抛弃了。

5.5.2 图像压缩的基础

数据压缩技术有多种不同的分类方法,其中一种是按压缩技术所依据和使用的数学理论和计算方法进行分类,这时可将压缩技术分为:

(1) 统计编码(Statistical Coding)。

(2) 预测编码(Predictive Coding)。

(3) 变换编码(Transform Coding)。

根据解码后的数据与原始数据是否完全一致可分为:

(1) 无失真编码。要求在解码后得到的图像与原始图像严格相同,如改进的 Huffman(霍夫曼)编码。

(2) 有失真编码。该方法的还原图像较原始图像存在一定的误差,但视觉效果一般是可以接受的。

下面介绍压缩中的基础知识。

1. 熵(Entropy)

数据压缩不仅起源于 20 世纪 40 年代由 Claude Shannon 首创的信息论,

而且其基本原理即信息究竟能被压缩到多小，至今依然遵循信息论中的一条定理，这条定理借用了热力学中的名词“熵”来表示一条信息中真正需要编码的信息量。

用0和1组成的二进制数码为含有 n 个符号的某条信息编码，假设符号 $F(n)$ 在整条信息中重复出现的概率为 $P(n)$，则该符号的熵(即表示该符号所需的位数)为：

$$E(n) = -\log_2(P(n)) \tag{5.36}$$

整条信息的熵(即表示整条信息所需的位数)为：

$$E = \sum E(n)$$

例如，字符串 aabbaccbaa，这个字符串中只有 a、b、c 三个字符。字符串长度为10，字符 a、b、c 分别出现了5、3、2次，则 a、b、c 在信息中出现的概率分别为0.5、0.3、0.2，它们的熵分别为：

$E(\mathrm{a}) = -\log_2(0.5) = 1$

$E(\mathrm{b}) = -\log_2(0.3) = 1.732$

$E(\mathrm{c}) = -\log_2(0.2) = 2.322$

整条信息的熵(即表达整个字符串需要的位数)为：

$$E = E(\mathrm{a}) \times 5 + E(\mathrm{b}) - 3 + E(\mathrm{c}) \times 2 = 14.855(\text{位})$$

如果用计算机中常用的 ASCII 编码表示上面的字符串，则需要整整80位。从这里可以看出为什么信息能被压缩而不丢失原有的信息内容。数据压缩的基本原则就是用较少的位数表示频繁出现的符号。

2. 率失真函数理论

在有失真的压缩中，为确保还原后的数据能基本保存原数据的特征，丢弃部分信息造成的失真应限制在某个规定的范围内；从另一角度讲，这种把失真限制在某一允许限度内，可使图像编码达到最高的压缩比的压缩编码，又可称为率失真编码或限失真编码。

率失真理论(Rate Distortion Theory)的目的是寻找一种联系定字长的编码策略的失真度(Distortion，重构的误差)和编码时的数据率(Data Rate，例如，每像素的位数)的方法。由于该理论假定输入图像是连续的，所以在有限数据率的条件下，由于存在量化误差，失真度永远不为零。虽然率失真函数理论没有确定最优的编码器，但是它为达到最佳效果提供了一些条件。

当使用有失真压缩方法时，重构的图像 $g(x,y)$ 将与原始图像 $f(x,y)$ 有所不同。定义失真度来表示不同的程度：

$$D = E\{[f(x,y) - g(x,y)]^2\} \tag{5.37}$$

失真度可以由重构的均方误差来定量确定。

如果定义一个最大容许失真度 D^0，那么编码时对应的比特率的下限 $R(D^0)$为 D^0 的单调递减函数。$R(D^0)$称为率失真函数，其函数图如图 5.39 所示。

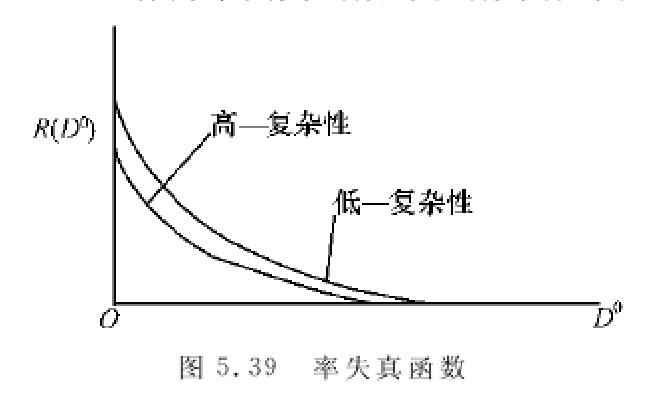


图 5.39　率失真函数

率失真函数具有以下两个特点：

(1) 任何指定的失真度 D，都可以找到一个速率任意接近 $R(D)$的编码方案，而平均失真也任意趋近于 D。

(2) 如果用低于 $R(D)$的速率进行编码，就不可能找到一个失真度为 D 或优于 D 的码。

3. 模型

从压缩的原则可知：要压缩信息，首先要分析清楚信息中每个符号出现的概率。不同的压缩程序通过不同的方法确定符号的出现概率，对符号的概率计算地越准确，也就越容易得到好的压缩效果。在压缩程序中，用来处理输入信息，计算符号的概率并决定输出哪个或哪些代码的模块叫做模型。

基于对每个字符出现次数的统计得到字符概率的模型可以统称为“统计模型”，统计模型有两类。

(1) 静态统计模型。预先扫描文件中的所有字符，统计出每个字符出现的概率，这种方法在压缩术语里叫做“静态统计模型”。但是，不同的文件中，字符有不同的分布概率，要么先花上大量的时间统计要压缩的所有文件中的字符概率，要么为每一个单独的文件保存一份概率表以备解压缩时需要。糟糕的是，不但扫描文件要消耗大量时间，而且保存一份概率表也使压缩后的文件增大了不少。所以，在实际应用中，“静态统计模型”应用得很少。

(2) 自适应模型。真正的压缩程序中使用的大多是一种叫“自适应模型”的东西。自适应模型可以说是一台具有学习功能的自动机。它在信息被输入之前对信息内容一无所知并假定每个字符的出现概率均等，随着字符不断被输入和编码，它统计并记录已经出现过的字符的概率并将这些概率应用于对后续字符的编码。也就是说，自适应模型在压缩开始时压缩效果并不理想，但随着压缩的进行，它会越来越接近字符概率的准确值，并达到理想的压缩效果。自适应模型还可以适应输入信息中字符分布的突然变化，可以适应不同文件中的字符分布而不需要保存概率表。

另一大类模型叫做“字典模型”。例如，在生活中提到“北大”这个词的时候，人

们都知道其意思是指“北京大学”,类似的例子还有不少,但前提是人们心中都有一本约定俗成的缩写字典。字典模型也是如此,它并不直接计算字符出现的概率,而是使用一本字典,随着输入信息的读入,模型找出输入信息在字典中匹配的最长的字符串,然后输出该字符串在字典中的索引信息。匹配越长,压缩效果越好。事实上,字典模型本质上仍然是基于对字符概率的计算,只是字典模型使用整个字符串的匹配代替了对某一字符重复次数的统计。字典模型得到的压缩效果仍然无法突破熵的极限。

对通用的压缩程序来说,保存一本大字典所需的空间仍然是无法让人忍受的,况且,任何一本预先定义的字典都无法适应不同文件中数据的变化情况。字典模型也有相应的“自适应”方案。人们可以随着信息的不断输入,从已经输入的信息中建立合适的字典,并不断更新这本字典,以适应数据的不断变化。

4. 编码

模型确定了对某一个符号该用多少位二进制数进行编码,现在的问题是,如何设计一种编码方案,使其尽量精确地用模型计算出来的位数表示某个符号。

如果对 a 用三个二进制位就可以表示,而对 b 用四个二进制位就可以表示,那么,在解码时,面对一连串的二进制流,怎么知道哪三个位是 a,哪四个位是 b 呢?所以,必须设计出一种编码方式,使解码程序可以方便地分离每个字符的编码部分,于是有了一种叫“前缀编码”的技术。该技术的主导思想是,任何一个字符的编码,都不是另一个字符编码的前缀。反过来说就是,任何一个字符的编码,都不是由另一个字符的编码加上若干位 0 或 1 组成。下面是前缀编码的一个最简单的例子:

符号	编码
A	0
B	10
C	110
D	1110
E	11110

有了上面的码表,就可以轻松地从下面这串二进制流中分辨出真正的信息内容了:

1110010101110110111100010——DABBDCEAAB

变长编码是统计编码中最主要的一种方法。变长编码的目标就是使平均码长达到最低。但是,这种最优必须在一定的限制下进行。编码的基本限制就是码字要有单义性和非续长性。

单义性代码:任意一个有限长的码字序列只能被分割成一个一个的码字,而任何其他分割方法都会产生一些不属于码字集合中的码字。符合这个条件的代码就叫单义代码。

非续长代码:任意一个码字都不是其他码字的续长。换句话说,就是码字集合中的任意一个码字都不是由其中一个码字在后面添上一些码元构成的。很容易看出非续长代码一定是单义的,但是,单义代码却不一定是非续长的。

表5.6列出了四种代码。

表5.6 四种代码表

信源	概率	码Ⅰ	码Ⅱ	码Ⅲ	码Ⅳ
u_1	0.5	0	0	0	0
u_2	0.25	0	1	10	01
u_3	0.125	1	00	110	011
u_4	0.125	10	11	111	0111

对码Ⅰ来说,如果在接收端收到0就无法判断是 u_1 还是 u_2,因此,在接收端不能正确译码,显然码Ⅰ缺乏单义可译性。码Ⅱ也有重要缺陷,例如,发送端发出 u_1u_2 这样一个序列,其码字将是00,但是在接收端既可判作 u_1u_1,也可以判作 u_3,所以这种码也缺乏单义可译性。而且,可以看出 u_3 的代码00是在 u_1 的代码0后面又加上一个0得到的,因此,又是可续长的。显然,码Ⅱ也不能使用。而码Ⅲ既具备单义可译性又是非续长的码,所以它是可用的。码Ⅳ也具有单义可译性,但是却缺乏非续长性。例如,当收到0时,不能立刻判断它是 u_1,必须等第二个码字出现,如果第二个码字是0则可以判断第一个码是 u_1,当第二个码字是1时,又无法判断了,还必须等待下一个码字出现才又可能判断,所以,续长码是无法及时译码的。

从上面可以看出,使平均码长最短的码只是在单义可译性和非续长性的约束下才能有意义。至于变长码的存在定理以及平均码长的最低限是否存在等问题,在信息论中都有详细的定理加以证明及讨论,在此不加赘述了。

5.5.3 压缩编码

高效编码的主要方法是尽可能去除信源中的冗余成分,从而以最少的数码率传递最大的信息量。冗余度存在于像素间的相关性及像素值出现概率的不均等性之中。对于有记忆性信源来说首先要去除像素间的相关性,从而达到压缩数码率的目的。对于无记忆性信源来说,像素间没有相关性,可以利用像素灰度值出现概率的不均等性,采用某种编码方法,也可以达到压缩数码率的目的。下面按照无失真编码和有失真编码的分类介绍常用的编码方法。

1. 行程编码

有些图像,尤其是计算机生成的图形往往有许多颜色相同的图块。在这些图块中,许多连续的扫描行都具有同一种颜色,或者同一扫描行上有许多连续的像素都具有相同的颜色值。在这些情况下就可以不需要存储每一个像素的颜色值,而

仅仅存储一个像素值以及具有相同颜色的像素数目。这种编码称为行程编码，或称游程编码，常用 RLE(Run－Length Encoding)表示。

这种压缩编码技术相当直观和经济，运算也相当简单，因此解压缩速度很快。RLE 的压缩率的大小取决于图像本身的特点。如果图像中具有相同颜色的横向色块越大且这样的图像块数目越多，压缩比就越大；反之就越小。如果图像中有大量纵向色块，则可先把图像旋转 90°，再用 RLE 压缩，也可以得到较大的压缩比。

RLE 压缩编码尤其适用于计算机生成的图形图像，对减少存储容量很有效。然而，对自然图像来说就完全不同。由于自然图像的颜色往往是五光十色，它的行程长度非常短，若用 RLE 对它进行编码，不仅不能把图像数据压缩，反而越压越多，要用更多的代码来表示。因此对复杂的图像都不能单纯地采用 RLE 进行编码。

2. 预测编码

预测编码即去除相邻像素之间的相关性和冗余性，只对新的信息进行编码。举个简单的例子，因为像素的灰度是连续的，所以在一片区域中，相邻像素之间灰度值的差别可能很小，如果只记录第一个像素的灰度，其他像素的灰度都用它与前一个像素灰度之差来表示，就能起到压缩的目的。如 248，2，1，0，1，3，实际上这 6 个像素的灰度是 248，250，251，251，252，255。表示 250 需要 8 个比特，而表示 2 只需要 2 个比特，这样就实现了压缩。

常用的预测编码有 Δ 调制(Delta Modulation，DM)、微分预测编码(Differential Pulse Code Modulation，DPCM)，具体的细节在此就不详述了。

3. 霍夫曼(Huffman)编码

霍夫曼编码(Huffman Coding)最早于 20 世纪 50 年代提出，它是一种无损的统计编码方法。它采用变长的码来使冗余量达到最小，用一棵二叉树来编码，使常出现的字符用较短的码代表，不常出现的字符用较长的码表示。静态霍夫曼编码使用一棵在压缩之前就建好的编码树，它是根据可能的字符出现的概率表来生成的。相反，动态霍夫曼编码是在编码过程中建立它的编码树。霍夫曼码编码使用以下步骤：

(1) 首先统计出每个符号出现的概率，然后根据符号概率的大小按由大到小顺序对符号进行排序，见表 5.7；

(2) 每一次选出概率最小的两个值，作为二叉树的两个叶子节点，将概率之和作为它们的根节点，如图 5.40 中节点 P1，这两个叶子节点不再参与比较，新的根节点参与比较；

(3) 重复步骤(2)，得到节点 P2、P3 和 P4，形成一棵“树”，其中的 P4 称为根节点；

(4) 从根节点 P4 开始到相应于每个符号的“树叶”，从上到下标上“0”(上枝)或者“1”(下枝)，至于哪个为“1”哪个为“0”则无关紧要，最后的结果仅仅是分配的代码不同，而代码的平均长度是相同的；

(5) 从根节点 P4 开始顺着“树枝”到每个“叶子”分别写出每个符号的代码，见表 5.7；

表 5.7　霍夫曼编码举例

符号	出现的次数	$\log \frac{1}{P_i}$	分配的代码	需要的位数
A	15(0.3846)	1.38	0	15
B	7(0.1795)	2.48	100	21
C	6(0.1538)	2.70	101	18
D	6(0.1538)	2.70	110	18
E	5(0.1282)	2.96	111	15

(6) 图 5.40 所示图像的熵为：

$$H(S) = \frac{15}{39} \times \log \frac{39}{15} + \frac{7}{39} \times \log \frac{39}{7} + \cdots + \frac{5}{39} \times \log \frac{39}{5} = 2.1859$$

霍夫曼编码的码长虽然是可变的，但却不需要另外附加同步代码。例如，码串中的第 1 位为 0，那么肯定是符号 A，因为表示其他符号的代码没有一个是以 0 开始的，因此下一位就表示下一个符号代码的第 1 位。同样，如果出现“110”，那么它就代表符号 D。如果事先编写出一本解释各种代码意义的“词典”，即码簿，就可以根据码簿一个码一个码地依次进行译码。

采用霍夫曼编码时要注意以下两个问题。

(1) 霍夫曼编码没有错误保护功能，会产生错误传播(Error Propagation)。在译码时，如果码串中没有错误，就能一个接一个地正确译出代码；但如果码串中有错误，哪怕仅仅其中 1 位出现错误，不但这个码本身译错，这个码后面的译码也会产生错误。计算机对这种错误也无能为力，不能纠正这种错误。

(2) 霍夫曼编码是可变长度编码，因此很难随意查找或调用压缩文件中间的内容。

虽然有以上两个缺点，但是霍夫曼编码还是得到了广泛应用。

4. 有失真编码方法

有失真编码方法的还原图像较原始图像存在一定的误差，但视觉效果一般是

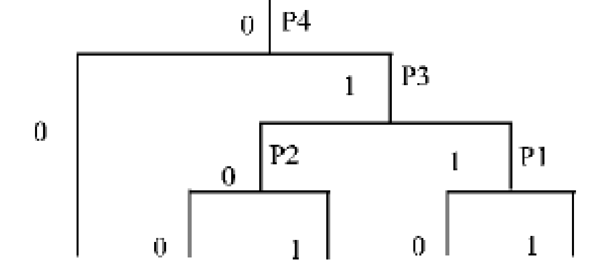


图 5.40　霍夫曼编码方法

可以接受的。根据有失真编码的原理进行分类，可以有变换编码、量化编码、信息熵编码、分频带编码、结构编码及基于知识的编码等。

（1）量化与向量量化编码的本质也是针对统计冗余进行压缩，不过从表现形式上看，好像是无关的。

（2）信息熵编码是根据信息熵原理，让出现概率大的用短的码字表达，反之用长的码字表达，最常见的如 Huffman 编码、行程编码以及算术编码等。

（3）分频带编码是将图像数据变换到频域后，按频率分频带，然后用不同的量化器进行量化，从而达到最优的组合。或者是分步渐进编码，即开始对某一频带的信号进行解码，然后扩展到所有频带。随着解码数据的增加，解码图像也逐渐清晰起来。此方法对于远距离图像模糊查询与检索比较有效。

（4）结构编码也称为第二代编码。编码时首先将图像中的边界、轮廓、纹理等结构特征求出来，然后保存这些参数信息。解码时，根据结构和参数信息进行反合成，从而恢复图像。

（5）基于知识的编码主要用于可规则描述的图像，如人脸，利用人们对人脸的知识形成一个规则库，据此将人脸的变换用一些参数进行描述，从而用参数与模型就可以实现人脸的图像编码与解码。

（6）变换编码也是一种针对统计冗余进行压缩的方法。所谓变换编码是将时域图像（空间）变换到频域上进行处理的方法。因为由时域映像到频域总是通过某种变换进行的，所以称为变换编码方法。在空间上具有强相关的信号，反映在频域上是在某些持定的区域中能量集中在一起，或者是系数矩阵的分布具有某种规律，这就可以利用这些规律分配频域上的量化比特数，从而达到压缩的目的。常用的变换有 KL 变换，DCT、DST、DFT、Haar 变换、walsh－Hadamard 变换以及用途广泛的小波变换等。变换编码有两个最明显的特点：一是可以得到高的压缩比；二是比预测等其他方法的计算复杂性高。在变换后，由于在频域上信息是按照频谱的能量与频率分布排列的，只要对频域平面量化器进行合理的（非均匀）比特分配，高能量区给以高的比特数，低能量区给以低的比特数，就可以得到高的压缩能力。

5.5.4 图像压缩的 Matlab 实现

1．离散余弦变换在图像压缩中的应用

离散余弦变换在图像压缩中具有广泛的应用，它是 JPEG、MPEG 等数据压缩标准的重要数学基础。

和相同图像质量的其他常用文件格式（如 GIF（可交换的图像文件格式）、TIFF（标签图像文件格式）、PCX（图形文件格式））相比，JPEG 是目前静态图像中压缩比最高的。JPEG 比其他几种压缩比要高得多，而图像质量都差不多（JPEG 处理的颜色只有真彩色和灰度图）。正是由于其高压缩比，使得 JPEG 被广泛地应

用于多媒体和网络程序中。JPEG 有几种模式，其中最常用的是基于 DCT 的顺序型模式，又称为基本系统(Baseline)，以下都针对这种格式进行讨论。

DCT 压缩的过程为：

(1) 首先将输入图像分解为 8×8 或 16×16 的块，然后对每个子块进行二维的 DCT；

(2) 将变换后得到的量化的 DCT 系数进行编码和传送，形成压缩后的图像格式。

DCT 解压的过程为：

(1) 对每个 8×8 或 16×16 块进行二维离散余弦逆变换；

(2) 将反变换的矩阵的块合成一个单一的图像。

从 5.2 节可以看到余弦变换具有把高度相关数据能量集中的趋势，DCT 后矩阵的能量集中在矩阵的左上角，右下方的大多数的 DCT 系数值非常接近于 0。对于通常的图像来说，舍弃这些接近于 0 的 DCT 的系数值，并不会对重构图像的画面质量带来显著下降。所以，利用 DCT 进行图像压缩可以节约大量的存储空间。压缩应该在最合理地近似原图像的情况下使用最少的系数。使用系数的多少也决定了压缩比的大小。

在压缩过程的第 2 步中，可以合理地舍弃一些系数，从而得到压缩的目的。在压缩过程第 2 步的时候，还可以采用 RLE 和 Huffman 编码来进一步压缩。

下面用上述方法对一幅图像分成 8×8 的块进行压缩。

【例 5.20】把输入图像 cameraman.tif 划分成 8×8 的图像块，计算它们的 DCT 系数，并且只保留 64 个 DCT 系数中的 10 个。然后对每个图像块利用这 10 个系数进行离散余弦逆变换来重构图像。

```
I=imread('cameraman.tif');
I=im2double(I);
T=dctmtx(8);          %产生二维 DCT 矩阵
%计算二维 DCT,T 和 T 转置是 DCT 函数 P1 * x * P2 的参数
B = blkproc(I,[8 8],'P1 * x * P2',T,T');
%二值掩模,用来压缩 DCT 系数,只留下 DCT 系数中左上角的 10 个
mask =[1   1   1   1   0   0   0   0
       1   1   1   0   0   0   0   0
       1   1   0   0   0   0   0   0
       1   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0];
```

```
B2 = blkproc(B,[8 8],'P1. * x',mask);         %只保留 DCT 的 10 个系数
I2 = blkproc(B2,[8 8],'P1 * x * P2',T',T);    %重构图像
subplot(1,2,1);
imshow(I);title('原始图像');
subplot(1,2,2);
imshow(I2);title('压缩图像');
```

其显示结果如图 5.41 所示。



图 5.41　DCT 压缩图像示例

对比原始图像和压缩后的图像，虽然舍弃了 85%的 DCT 系数，但图像仍然清晰(当然有一些质量损失)。

2．小波分析用于图像压缩

小波分析用于信号与图像压缩是小波分析应用的一个重要方面。它的特点是压缩比高，压缩速度快，压缩后能保持信号与图像的特征基本不变，且在传递过程中可以抗干扰。

基于小波分析的图像压缩方法很多，比较成功的有小波包最好基方法、小波域纹理模型方法、小波变换零树压缩、小波变换向量量化压缩等。

【例 5.21】利用小波分析进行图像压缩。

说明：一个图像作小波分解后，可得到一系列不同分辨率的子图像，不同分辨率的子图像对应的频率是不相同的。高分辨率(即高频)图像上大部分点的数值都接近于 0，越是高频这种现象越明显。对一个图像来说，表现一个图像最主要的部分是低频部分，所以一个最简单的压缩方法是利用小波分解，去掉图像的高频部分而只保留低频部分。图像压缩可按如下过程进行处理。

```
clear;                      %清除 Matlab 工作环境中现有的变量
load wbarb;                 %装入图像
%显示原始图像
subplot(2,2,1);image(X);colormap(map);
```

```
title('原始图像');
disp('原始图像 X 的大小:');
whos('X');
%对图像用 bior3.7 小波进行二层小波分解
[c,s] = wavedec2(X,2,'bior3.7');
%提取小波分解结构中第一层的低频系数和高频系数
ca1 = appcoef2(c,s,'bior3.7',1);
ch1 = detcoef2('h',c,s,1);
cv1 = detcoef2('v',c,s,1);
cd1 = detcoef2('d',c,s,1);
%分别对各频率成分进行重构
a1=wrcoef2('a',c,s,'bior3.7',1);
h1=wrcoef2('h',c,s,'bior3.7',1);
v1=wrcoef2('v',c,s,'bior3.7',1);
d1=wrcoef2('a',c,s,'bior3.7',1);
c1=[a1,h1;v1,d1];
%显示分解后各频率分量的信息
subplot(2,2,2);image(c1);
axis square;
title('分解后低频和高频信息');
%下面进行图像压缩处理
%保留小波分解第一层低频信息,进行图像的压缩
%第一层的低频信息即 cal,显示第一层的低频信息
%首先对第一层信息进行量化编码
ca1=appcoef2(c,s,'bior3.7',1);
ca1= wcodemat(ca1,440,'mat',0);
%改变图像的高度
ca1=0.5 * ca1;
subplot(2,2,3);image(ca1);colormap(map);
axis square;
title('第一次压缩图像');
disp('第一次压缩图像的大小:');
whos('ca1');
%保留小波分解第二层低频信息,进行图像的压缩,此时压缩比更大
%第二层的低频信息即 ca2,显示第二层的低频信息
ca2=appcoef2(c,s,'bior3.7',2);        %进行小波二层分解
ca2= wcodemat(ca2,440,'mat',0);
%首先对第二层信息进行量化编码
```

```
ca2=0.25 * ca2;
subplot(2,2,4);image(ca2);colormap(map);
axis square;
title('第二次压缩图像');
disp('第二次压缩图像的大小:');
whos('ca2');
```

在工作窗口中得到以下结果：

```
原始图像 X 的大小:
Name        Size          Bytes   Class
X           256x256       524288  double array
Grand total is 65536 elements using 524288 bytes
第一次压缩图像的大小
Name        Size          Bytes   Class
ca1         135x135       145800  double array
Grand total is 18225 elements using 145800 bytes
第二次压缩图像的大小
Name        Size          Bytes   Class
ca2         75x75         45000   double array
Grand total is 5625 elements using 45000 bytes
```

其显示结果如图 5.42 所示。

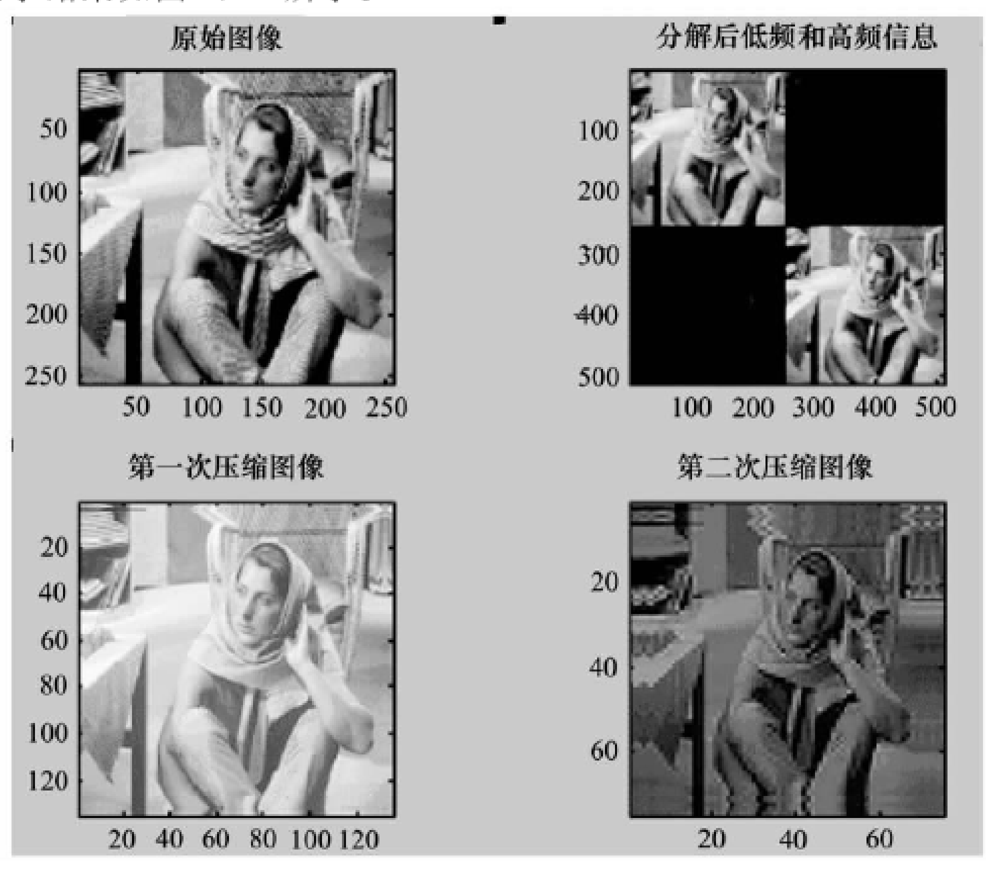


图 5.42　小波变换对图像的压缩

第6章 Matlab图像增强

6.1 图像增强原理及方法

对于一个图像处理系统来说,可以将处理分为三个阶段,在获取原始图像后,首先是图像预处理阶段,第二是特征抽取阶段,第三是识别分析阶段。图像预处理阶段尤为重要,如果这阶段处理不好,后面的工作根本无法展开。图像增强是图像预处理中重要的方法。

在实际应用中,系统获取的原始图像不是完美的,影响系统图像清晰程度的因素很多,例如,室外光照度不够均匀就会造成图像灰度过于集中;由CCD(摄像头)获得的图像经过A/D转换(数/模转换,该功能在图像系统中由数字采集卡来实现)、线路传送都会产生噪声污染等。因此图像质量不可避免地降低了,轻者表现为图像不干净,难于看清细节;重者表现为图像模糊不清,连概貌也看不出来。因此,在对图像进行分析之前,图像预处理中必须要对图像质量进行改善,一般情况下改善的方法有两类:图像增强和图像复原。图像增强不考虑图像质量下降的原因,只将图像中感兴趣的特征有选择地突出,而衰减不需要的特征,它的目的主要是提高图像的可懂度。图像增强的方法分为空域法和频域法两类,空域法主要是对图像中的各个像素点进行操作;而频域法是在图像的某个变换域内,修改变换后的系数,如傅里叶变换、DCT等的系数,对图像进行操作,然后再进行反变换得到处理后的图像。图像复原技术与增强技术不同,它需要了解图像质量下降的原因,首先要建立"降质模型",再利用该模型恢复原始图像。本章主要讨论图像增强。

图像增强技术的主要目标是,通过对图像的处理,使图像比处理前更适合一个特定的应用。在显示、打印、印刷、识别、分析、艺术创造等地方都可以应用到图像增强。

在图像增强中可能进行如下处理:去除噪声、边缘增强、提高对比度、增加亮度、改善颜色效果、改善细微层次等——通常与改善视觉效果相一致。

图像增强的主要目的有两个:一是改善图像的视觉效果,提高图像成分的清晰度;二是使图像变得更有利于计算机处理,如锐化处理可以突出图像边缘轮廓线,这样可以编程控制计算机进行跟踪,便可以作各种特征分析。

图像增强理论(方法)目前尚无统一的权威性定义,因为还没有衡量图像质量

的通用标准。从增强处理的作用域出发,图像增强可以分为空域增强方法和频域增强方法两大类。

(1) 空域增强方法:直接在图像所在的空间进行处理,也就是在像素组成的空间里直接对像素进行操作。

(2) 频域增强方法:用第 5 章的图像变换方法将原来的图像空间中的图像以某种形式转换到其他空间中,然后利用该空间的特有性质方便地进行图像处理,最后再转换回原来的图像空间中,从而得到处理后的图像。

6.2 空域变换增强

空域增强方法又可以分为两类:

(1) 基于像素点:即每次处理是对图像的每个像素进行的,增强过程对每个像素的处理与其他像素无关;

(2) 基于模板:对图像的每次处理是对小的子图像(模板)进行的。

6.2.1 直接灰度调整

在空间域内对图像进行点运算是一种既简单又重要的图像处理技术,它能让用户改变图像上像素点的灰度值,这样通过点运算处理将产生一幅新图像。

简单地说,灰度调整就是指对图像上各个像素点的灰度值 x 按某个函数 $T(x)$ 变换到 y 。例如,为了提高图像的清晰度,需要将图像的灰度级整个范围或其中某一段 (A,B) 扩展或压缩到 (A',B') ;在需要显示出图像的细节部分等处都要求采用灰度变换方法。灰度变换有时又被称为图像的对比度增强或对比度拉伸。假定输入图像中的一个像素的灰度级为 Z ,经过函数 $T(Z)$ 变换后输出图像对应的灰度级为 Z' ,其中要求 Z 和 Z' 都要在图像的灰度范围之内。根据 $T(Z)$ 的形式,可以将灰度变换分为线性变换和非线性变换。具体应用中采用何种 $T(Z)$,需要根据变换的要求而定。为了选择一种合理的变换函数,首先应该对原始图像的像素灰度值有一个大概了解,然后根据像素的统计特征来确定需要的变换函数类型。Matlab 图像处理工具箱提供了多个函数以返回与构成图像的灰度相关的信息。

1. 像素选择

为了能够提供图像上指定的像素的信息,Matlab 图像处理工具箱提供了两个函数:pixval 和 impixel。下面对这两个函数分别说明。

(1) 函数 pixval。

功能:得到像素的数据值,并且能够显示两个像素间的欧几里得几何距离。

语法格式:

```
pixval on
pixval off
pixval
pixval(fig,option)
```

说明：

使用 pixval 函数前必须先显示图像，然后在命令窗口中输入 pixval('on')，则在被显示图像的底端出现一个黑色状态栏。当光标在图像上移动时，则在黑色状态栏中显示光标所在像素的坐标和该像素的颜色数据。

当在图像中点击鼠标并拖动时，在黑色条中将显示最初点击像素和当前像素间的几何距离。

当在命令窗口中输入 pixval('off')，则会退出当前的交互操作。

【例 6.1】用 pixval 测试像素的值。

输入如下命令：

```
imshow canoe.tif;
pixval('on');
```

显示图 6.1 所示窗口。

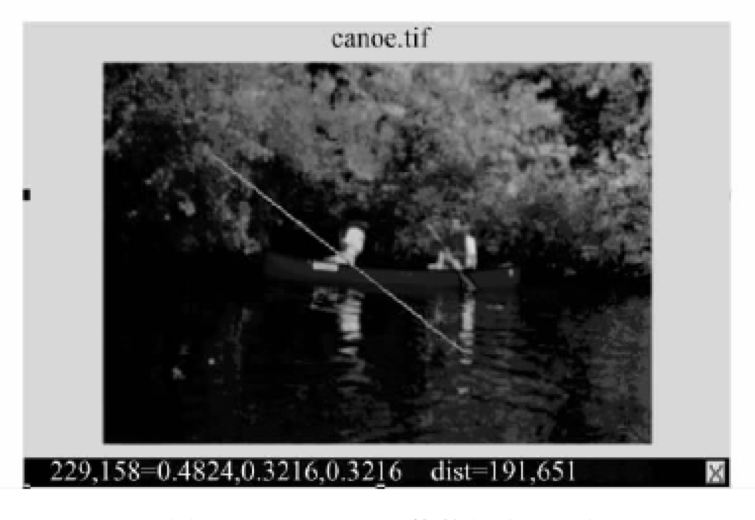


图 6.1 pixval 函数的运行界面

（2）函数 impixel。

功能：得到像素的数据。

语法格式：

```
[C,R,P]=impixel(X,MAP)
```

说明：

X 表示输入图像；MAP 为索引图像的调色板（仅当图像为索引图像时才有此参数）；*C* 表示指定像素的颜色；*R* 和 *P* 表示像素的坐标。

如果在输入图像参数后面给出指定像素坐标的向量，那么 impixel 函数将返回指定像素的颜色值。如果在输入图像参数后没有给出坐标值，则会进入交互方

式,根据鼠标在图像上单击的位置返回颜色值。

【例 6.2】用 impixel 函数测试图像上指定点的颜色数据。

```
RGB = imread('flowers.tif');
c = [12 146 410];
r = [104 156 129];
pixels = impixel(RGB,c,r)
```

两个被选像素的颜色数据如下:

```
pixels =
61     59     101
253    240    0
237    37     44
```

pixval 函数可以给出比 impixel 函数更多的像素信息,但是 impixel 函数能够将结果返回到一个变量中,以后可以通过交互式或非交互式的方法对这个变量进行访问或操作。

对于索引图像,pixval 和 impixel 函数都将显示调色板存储的 RGB 值,而不是调色板的索引。

2. 强度描述图

Matlab 图像处理工具箱提供了 improfile 函数用于得到图像中一条线段或多条线段强度(灰度)值,并绘制其图形。

函数:improfile。

功能:计算图像中一条线段或多条线段强度值。

语法格式:

```
C = improfile(I,xi,yi)
```

说明:

I 为输入图像;***xi*** 和 ***yi*** 是两个向量,用来指定线段的端点;*C* 是线段上各点的灰度或颜色。如果不指定 ***xi*** 和 ***yi*** 参数,则会进入交互式的处理模式。对于单独的线段,improfile 函数将会在二维视图中绘制点的灰度值;对于多条线段,improfile 函数将会在三维视图中绘制灰度值。

【例 6.3】用 improfile 函数显示两条线段的灰度值。

```
I = imread('alumgrns.tif');
subplot(1,2,1);imshow(I);
x = [35 338 346 103];
y = [253 250 17 148];
subplot(1,2,2);improfile(I,x,y), grid on
```

其显示结果如图 6.2 所示。

【例 6.4】用 improfile 函数绘制彩色图像的强度值。

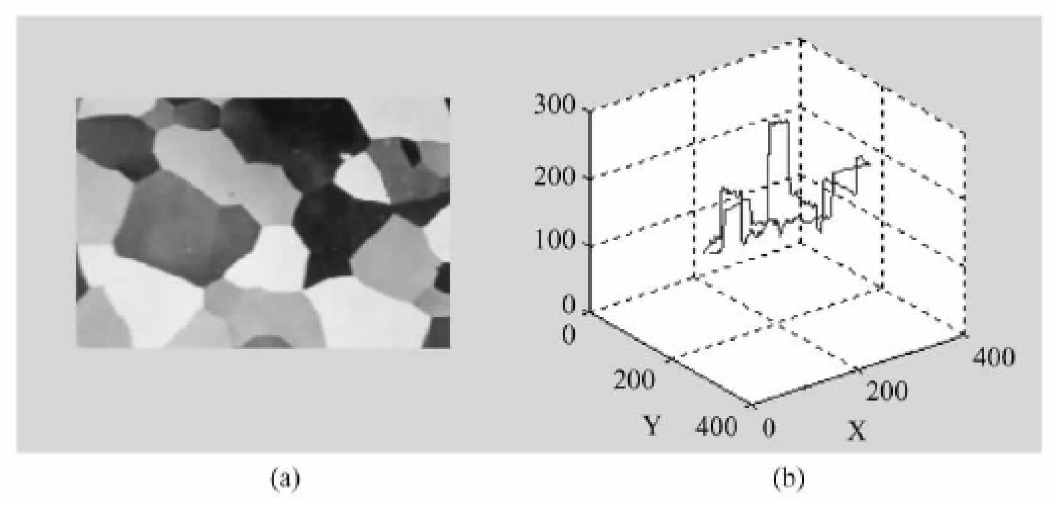


图 6.2 线段的灰度分布图像

(a)原图像;(b)两条线段的灰度分布三维显示。

绘制结果如图 6.3 所示。从图中可以看出,improfile 函数绘制的强度图是将红、蓝、绿三色分离的,各自均表达为独立的线条。

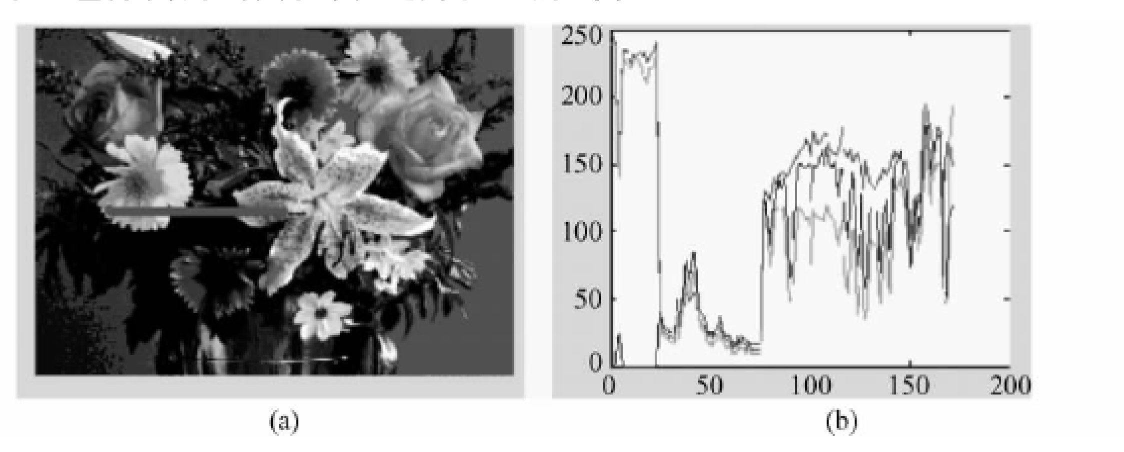


图 6.3 彩色图像的强度分布图像

(a)彩色图像;(b)线段强度的二维显示。

3.图像轮廓图

Matlab 图像处理工具箱提供了 imcontour 函数来显示灰度图像中数据的轮廓图,这个函数同 contour 函数非常相似,但 imcontour 函数功能更全。imcontour 函数能够自动设置坐标轴,使输出图像在其方向和纵横比上能够与显示的图像吻合。

【例 6.5】显示 rice.tif 的原始图像和图像的轮廓图。

```
I = imread('rice.tif');
subplot(1,2,1);imshow(I)
subplot(1,2,2);imcontour(I)
```

其显示结果如图 6.4 所示。

4.直方图

在数字图像处理中,一个最简单和最有用的工具是直方图,它概括了一幅图像的灰度级内容。任何一幅图像的直方图都包括了可观的信息,某些类型的图像还

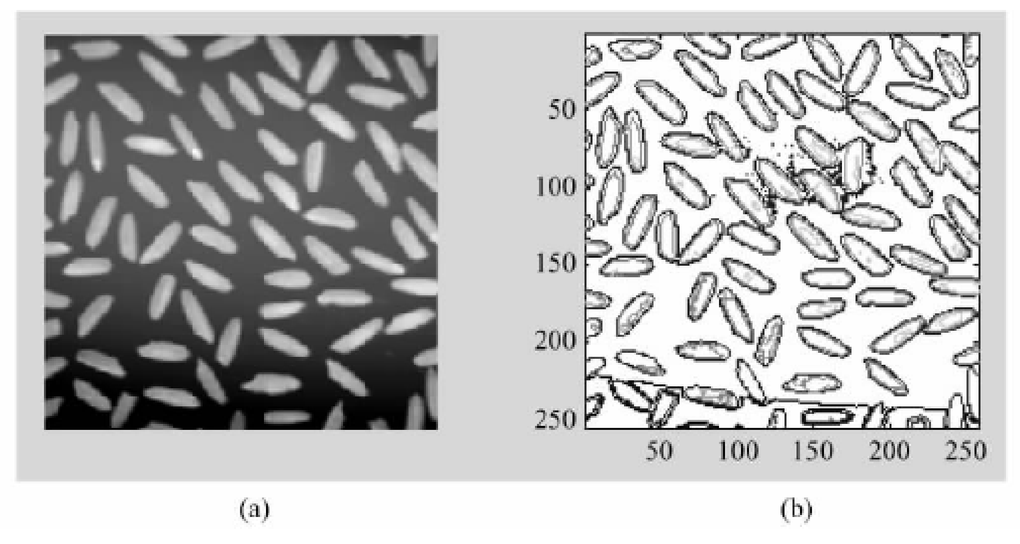


图 6.4　图像的轮廓图

(a)原始灰度图像;(b)图像的轮廓图。

可由其直方图完全描述。直方图的计算是简单的,特别是当一幅图从一个地方被复制到另一个地方时,直方图的计算可以用非常低的代价来完成。

灰度直方图是灰度级的函数,描述的是图像中具有该灰度级的像素的个数:其横坐标是灰度级,纵坐标是该灰度出现的频率(像素的个数)。Matlab 图像处理工具箱提供了 imhist 函数来显示图像的直方图。

函数:imhist。

功能:显示指定图像的直方图。

语法格式:

```
imhist(I,n)
imhist(X,map)
[counts,x] = imhist(...)
```

说明:

I 为输入图像;n 为指定的灰度级数目,默认值为 256;imhist(X,map)计算和显示索引色图像为 X 的直方图;map 为调色板;[counts,x] = imhist(...)返回直方图数据向量 counts 或相应的色彩值向量 $\boldsymbol{x}$。

【例 6.6】显示一个图像的直方图。

```
I = imread('pout.tif');
subplot(1,2,1);
imshow(I);
title('原图像');
axis square;
subplot(1,2,2);
imhist(I);
title('图像的直方图');
axis square;
```

得到显示的图像如图 6.5 所示。

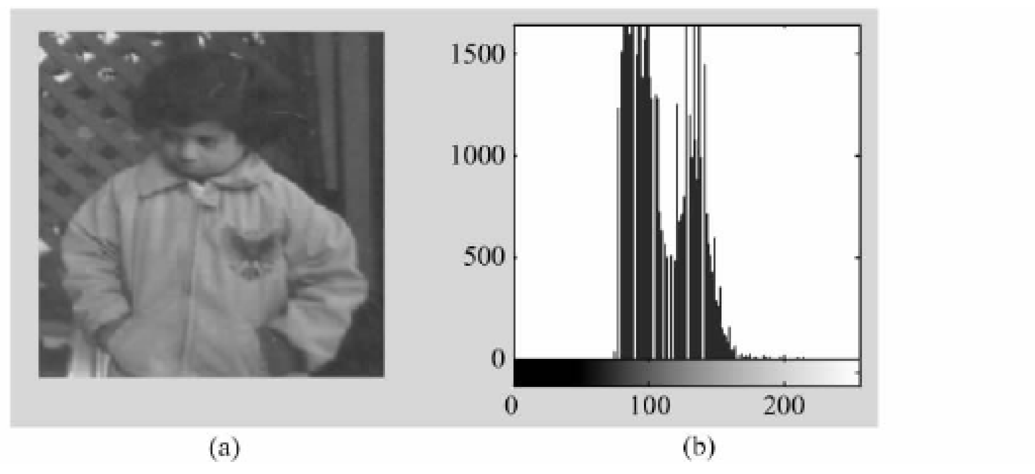


图 6.5 图像及其直方图显示

(a)原图像;(b)图像的直方图。

5. 对比度扩展(Contrast Stretching)以增强图像

假设有一幅图,由于成像时光照不足,使得整幅图偏暗(例如,整幅图像的灰度范围从 0 到 63);或者成像时光照过强,使得整幅图偏亮(例如,整幅图像的灰度范围从 200 到 255),这些情况称为低对比度,即灰度都在一个狭小的区域,没有分布到整个图像区域。灰度扩展的意思就是把灰度范围分布到整个灰度区域,使得该范围内的像素,亮的越亮,暗的越暗,从而达到增强对比度的目的。

以图 6.6 为例来进行说明。从图 6.6 的直方图可以看到:这个图的灰度过于集中在范围 45 到 200,导致图像对比度不够。可以用对比度扩展的方法增加图像的亮度。

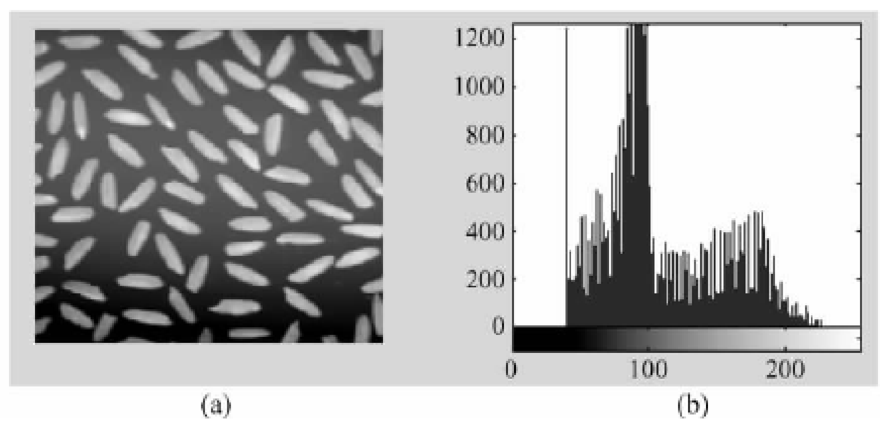


图 6.6 一幅较暗图像及直方图

(a)较暗图像;(b)直方图。

MATAB 工具箱提供了 imadjust 函数来实现图像的灰度级变换。

函数:imadjust。

功能:将图像的灰度值映像为一个新的数值范围。

语法格式:

```
J=imadjust(I,[low_in high_in],[low_out high_out],gamma)
newmap=imadjust(map,[low high],[bottom top],gamma)
```

说明：

I 是输入图像矩阵；*J* 是经过直方图变换后的输出图像矩阵；low_in 和 high_in 参数分别用来指定输入图像需要映像的灰度范围；low_out 和 high_out 指定输出图像所在的灰度范围；newmap＝imadjust(map,[low high],[bottom top],gamma)调整索引色图像的调色板 map。此时若[low high]和[bottom top]都是 2×3 矩阵，就根据它们的值分别调整 R、G、B 三个分量。

不管 *I* 属于哪一类，此处指定的强度值的范围均为[0,1]。如果 *I* 是 unint8，则会将用户指定的值乘以 255，然后将得到的结果作为实际的强度使用；如果是 unint16，则会乘以 65535。

gamma 是一个可选的参数。一般来说灰度间的映像是直线的，但是通过调整 gamma 参数可以变为非线性的映像。

【例 6.7】通过对比度扩展增强 rice.tif 的对比度。

从上面的分析可以看到，rice.tif 的灰度过于集中，对比度不够，即灰度没有低于 40 或者大于 200 的。所以，把图像的灰度映像到整个灰度级上，即 0 到 255，这样图形的对比度就大大提高了。

```
I = imread('rice.tif');
J = imadjust(I,[0.15 0.9],[0 1]);                    %进行直方图变换
subplot(1,2,1);
imshow(J);
subplot(1,2,2);
imhist(J);
```

其显示结果如图 6.7 所示。

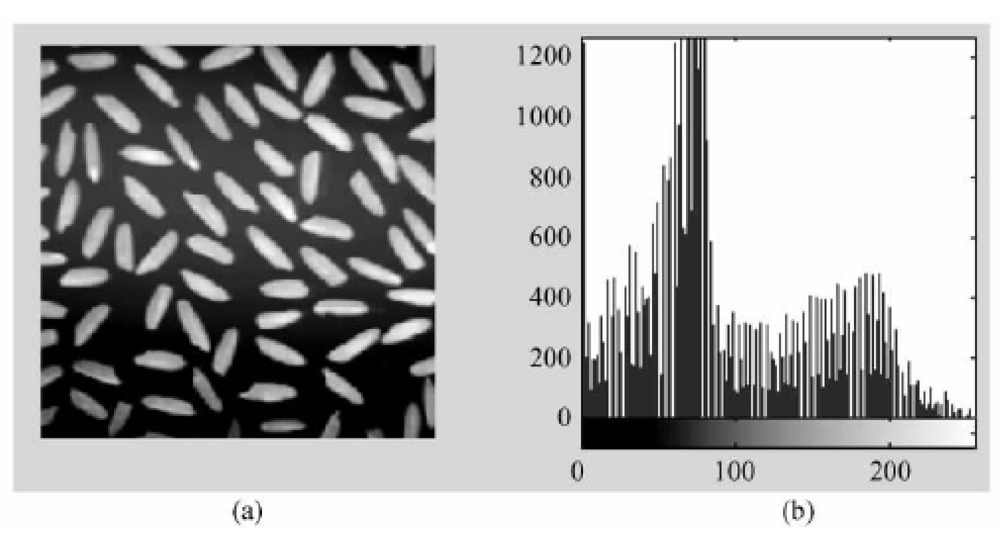


图 6.7　变换后的图像和直方图

(a)变换后的图像；(b)变换后图像的直方图。

从图 6.7 的结果图像和直方图可以看到：通过将图像从灰度范围[0.15,0.9]映像到[0,1]，图像的对比度明显增加。

直接的灰度变换曲线可以取得增强对比度的效果，对于低值灰度的图像，有时使用对数变换效果更好。对数变换可以扩展低值灰度，压缩高值灰度，这样可以使低值灰度的图像细节更容易看清楚。

对数变换的灰度映像采用如下表达式：

$$g(x,y) = \log(f(x,y)+1) \tag{6.1}$$

【例 6.8】采用对数变换来改善图像质量。

```
I=imread('pout.tif');
subplot(1,2,1);
imshow(I);
%对数运算不支持 unit8 类型，将图像转换为 double 类型
I=double(I);
J=log(I+1);                    %进行灰度的对数变换
subplot(1,2,2);
imshow(mat2gray(J));
```

其显示结果如图 6.8 所示。



图 6.8　用对数变换来增强图像

(a)原始图像；(b)对数变换后增强图像。

从图像对比可以看出，经过对数变换后窗格等细节更加清楚。

6.2.2　直方图处理

1. 直方图均衡化

直方图均衡化是一种最常用的直方图修正。它是把给定图像的直方图分布改造成均匀直方图分布，使输出像素灰度的概率密度均匀分布。用信息学的理论来解释，即具有最大熵(信息量)的图像为均衡化图像。直观地讲，直方图均衡化导致图像的对比度增加。

直方图均衡化的优点是能自动增强整个图像的对比度，但它的具体增强效果不容易控制，处理的结果总是得到全局均衡化的直方图。

直方图均衡化的基本思想是把原始图像的直方图变换成均匀分布的形式，这样就增加了图像灰度值的动态范围，从而达到了增强图像整体对比度的效果。具体方法是：

(1) 列出原始图像的灰度级 $S_k, k=0,1,\cdots,L-1$，其中 L 是灰度级的个数；

(2) 统计原始图像各灰度级的像素数目 n_k；

(3) 计算原始图像直方图各灰度级的频率数；

(4) 计算原始图像的累计直方图；

(5) 取整计算：

$$t_k = \mathrm{int}[(N-1)t_k + \frac{k}{N}]$$

(6) 确定映像关系：

$$S_k \to t_k$$

(7) 统计新直方图各个灰度级的像素 n_k；

(8) 计算新的直方图：

$$p_t(t_k) = n_k / N$$

Matlab 图像处理工具箱中提供了 histeq 函数来完成直方图均衡化。

函数：histeq。

功能：实现对输入图像的直方图均衡化

语法格式：

```
J = histeq(I,hgram)
J = histeq(I,n)
[J,T] = histeq(I,...)
newmap= histeq(X,map,hgram)
newmap= histeq(X,map)
[newmap,T] = histeq(X,...)
```

说明：

n 表示输出图像的灰度级数目，是一个可选参数，默认值是 64；J＝histeq(I, hgram)将原始图像 I 的直方图变成用户指定的向量 hgram，hgram 中的各元素值域为[0,1]；[J,T]＝histeq(I,n)返回从图像 I 的灰度直方图变换成图像 J 的直方图的变换 $\boldsymbol{T}$；newmap＝ histeq(X,map)用来对索引图像进行处理，索引图像的返回值 newmap 将是输出图像的调色板。

【例 6.9】对一个图像进行直方图均衡化并绘制直方图均衡化的转移函数的变换曲线。

```
I = imread('tire.tif');
J = histeq(I);                          %进行直方图均衡化
subplot(2,2,1);
```

```
imshow(I);
title('原始图像');                    %显示原始图像
subplot(2,2,2);
imshow(J);
title('直方图均衡图像');               %显示直方图均衡化后的图像
subplot(2,2,3);
imhist(I);
title('原始图像直方图');               %显示原始图像的直方图
subplot(2,2,4);
imhist(J);
title('均衡化图像直方图');             %显示均衡化后图像的直方图
I = imread('tire.tif');
[J,T] = histeq(I);                    %进行直方图均衡化
figure,plot((0:255/255),T);           %绘制转移函数的变换曲线
```

原始图像,原始直方图、处理后的图像和直方图如图 6.9 所示。从直方图来看,处理后的 tire.tif 的图像直方图分布更均匀了,在每个灰度级上图像都有像素点。从处理前后的图像可以看出,许多在原始图像中看不清楚的细节在直方图均衡化处理后所得到的图像中都变得十分清晰。直方图均衡化的转移函数的变换曲线如图 6.10 所示。

从图 6.9 可以看出,经过直方图均衡化处理后,图像变得清晰了,但是均衡化处理后的图像只是近似均匀分布。均衡化图像的动态范围扩大了,但其本质是扩大了量化间隔,而量化级别反而减少了,因此,直方图均衡化存在着以下三个缺点:

(1) 变换后图像的灰度级减少,某些细节消失。

(2) 某些图像,如直方图有高峰,经处理后对比度过分增强。

(3) 原来灰度不同的像素经处理后可能变得相同,形成了一片相同灰度的区域,各区域之间有明显的边界,从而出现了伪轮廓。

2. gamma 校正

在灰度变换函数 imadjust 函数中有一个可选参数 gamma,根据 gamma 值的不同,可以采用不同的非线性的映射方法将输入图像映射到输出图像。默认情况下为 1,即采用的是线性变换。gamma 因子的取值决定了映射的方式,即决定了对图像的变换是进行增强低灰度还是增强高灰度。gamma 因子的映射方式如图 6.11 所示。

在函数 J = imadjust(I,[low_in high_in],[low_out high_out],gamma)中,gamma 可以是 0 到无穷的任意数值。在默认情况下 gamma=1,表示在 low_in 和 high_in 之间的数值将会线性地映射为 low_out 和 high_out 之间的数值;当 gam-

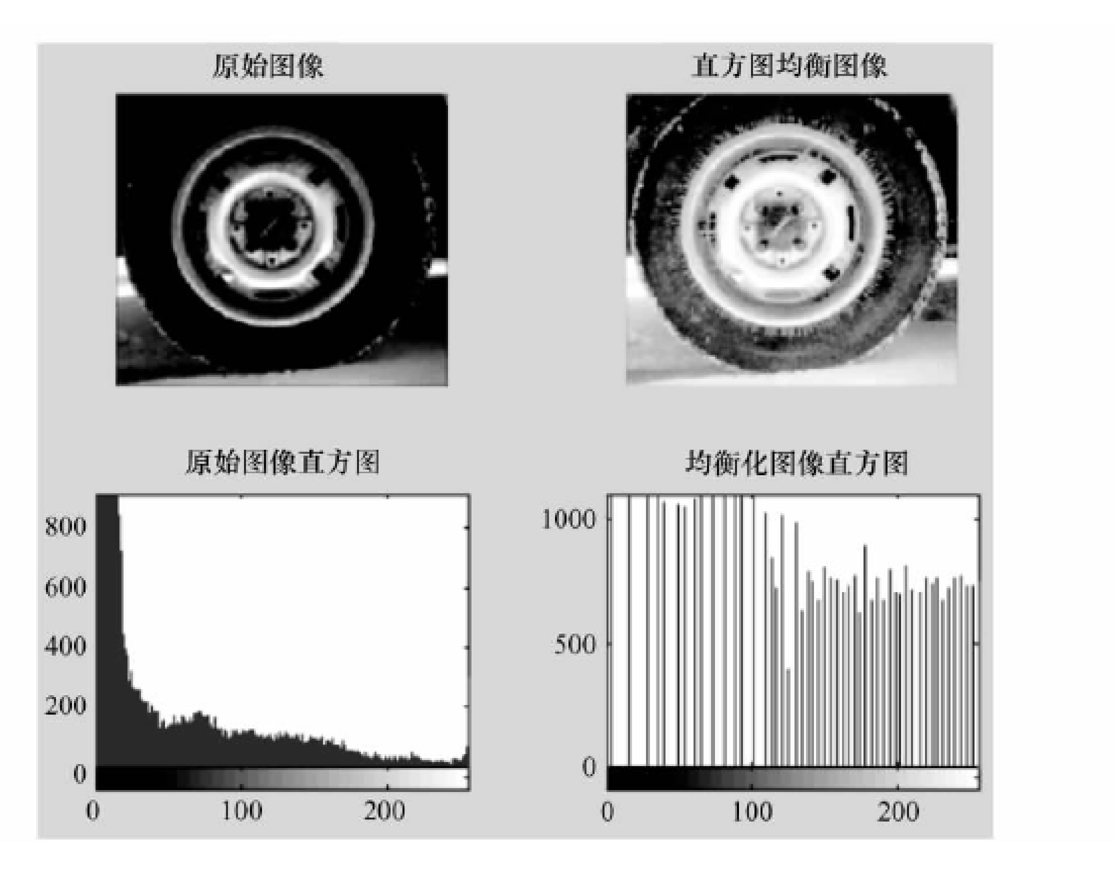


图 6.9　直方图均衡化图像的对比

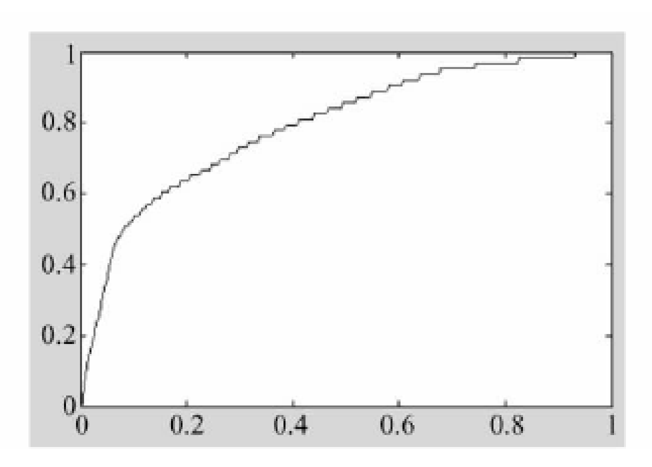


图 6.10　直方图均衡化的转移函数的变换曲线

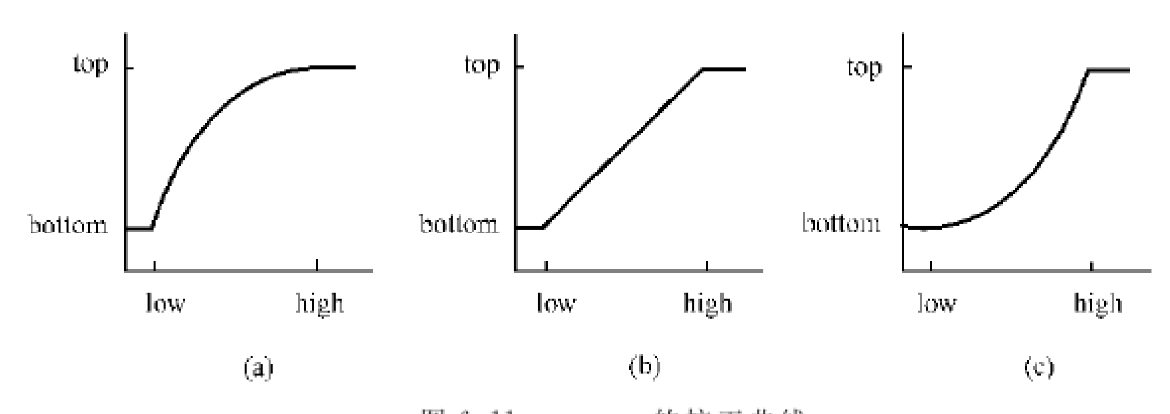


图 6.11　gamma 的校正曲线

(a)gamma<1 时的映射方式;(b)gamma=1 时的映射方式;(c)gamma>1 时的映射方式。

ma<1，映射将会通过对图像的像素灰度加权进行计算，使输出像素灰度值比原来大；当 gamma>1，映射将会通过对图像的像素灰度加权进行计算，使输出像素灰度值比原来小。

【例 6.10】对一幅图像采用不同的 gamma 进行变换。

```
[X,map] = imread('forest.tif')
I = ind2gray(X,map);                    %转变为灰度图像
J1 = imadjust(I,[],[],0.5);             %gamma 取值为 0.5 的变换
J2 = imadjust(I,[],[],1);               %gamma 取值为 1 的变换
J3 = imadjust(I,[],[],2);               %gamma 取值为 2 的变换
subplot(2,2,1);
imshow(I);title('原始图像');
subplot(2,2,2);
imshow(J1);title('gamma 为 0.5 的变换');
subplot(2,2,3);
imshow(J2);title('gamma 为 1 的变换');
subplot(2,2,4);
imshow(J3);title('gamma 为 2 的变换');
```

其显示结果如图 6.12 所示。

图 6.12　不同 gamma 取值变换图像的对比

gamma 指定了变换曲线的形状，描述了 I 和 J 的值之间的关系。从图 6.12 可以看出 gamma 参数取不同的值对输出图像的影响，通常当 gamma>1 时，图像变

暗，而当 gamma<1 时，图像变亮。

3. 直方图规定化

直方图均衡化的优点是能够自动增强整个图像的对比度，但它的具体增强效果不容易控制，处理的结果总是得到全局均匀化的直方图。但是在某些情况下，为了有选择地增强某个灰度值范围的对比度，输出图像的直方图是人为规定的，这时可以采用比较灵活的直方图规定化。

所谓直方图规定化，就是通过一个灰度映像函数 Gnew＝F(Gold)，将原灰度直方图改造成所希望的直方图。所以，直方图修正的关键就是灰度映像函数，前面介绍的对数变换等都是一种灰度函数映像。

直方图规定化方法主要有以下三个步骤。

(1) 对原始图像的直方图进行灰度均衡化：

$$t_k = EH_s(s_i) = \sum_{i=0}^{k} P_s(s_i) \qquad k = 0,1,\cdots,M-1$$

(2) 规定所需要的直方图，并计算能使规定的直方图均衡化的变换：

$$v_l = EH_u(u_j) = \sum_{j=0}^{l} P_u(u_j) \qquad j = 0,1,\cdots,N-1$$

(3) 将第一步所得的变换翻转过来，即将原始图像对应映射到规定的直方图，将所有的 $P_s(s_i)$ 对应到 $P_u(u_j)$ 上。

在 Matlab 中可以调用函数 J＝histeq(I,hgram)来实现直方图规定化。其中 hgram 是用户指定的向量，规定将原始图像 I 的直方图近似变换成 hgram，hgram 中的每一个元素都在[0,1]中。

【例 6.11】对图像 tire.tif 进行规定直方图的变换。

```
I=imread('tire.tif');
hgram=0:255;                    %直方图变换的规定化函数
J=histeq(I,hgram);              %将图像向指定的直方图变换
subplot(2,2,1)
imshow(I);title('原始图像');
subplot(2,2,2)
imshow(J);title('直方图规定化后图像');
subplot(2,2,3)
imhist(I);title('原始图像直方图');
subplot(2,2,4)
imhist(J);title('规定化后图像直方图');
```

其显示结果如图 6.13 所示。

和直方图均衡化的图像进行比较可以知道，直方图在高灰度值一侧更为密集，指定变化后的图像比直方图均衡化后的图像更亮，在较暗的区域细节更加清楚。

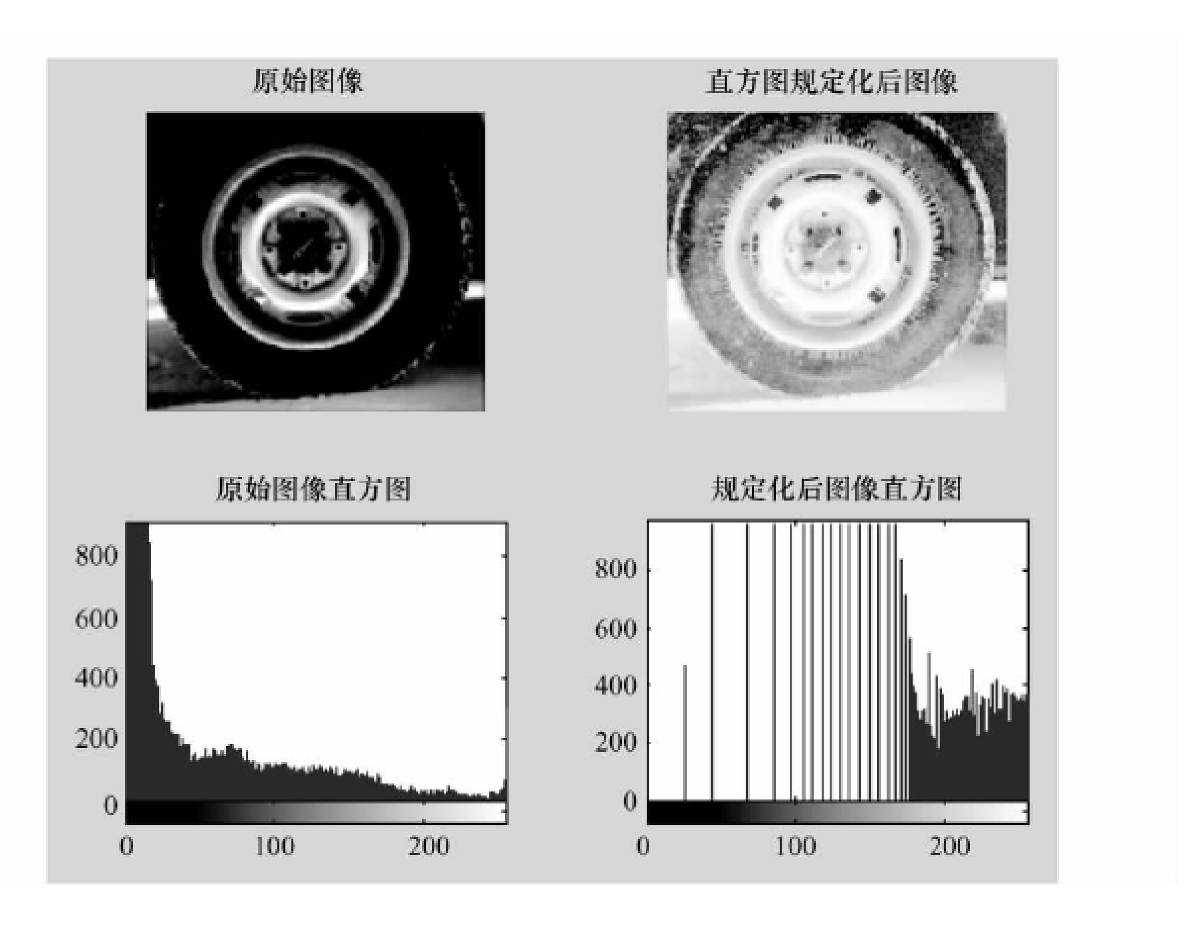


图 6.13　直方图规定化变换图像的对比

6.2.3　图像间的代数运算

代数运算也可以实现图像的增强。代数运算是指对两幅输入图像进行像素点对像素点的加、减、乘或除计算而得到输出图像的运算。对于相加和相乘的情形，可能不仅有两幅图像参加运算。在一般情况下，输入图像之一可能为常数，并且，加、减、乘、除常数可按线性的点运算来对待。当两幅输入图像完全相同时，也如此。

四种图像处理代数运算的数学表达式如下：

$$C(x,y) = A(x,y) + B(x,y)$$

$$C(x,y) = A(x,y) - B(x,y)$$

$$C(x,y) = A(x,y) \times B(x,y)$$

$$C(x,y) = A(x,y) \div B(x,y)$$

其中 $A(x,y)$ 和 $B(x,y)$ 为输入图像，而 $C(x,y)$ 为输出图像。还可通过适当的组合，形成涉及几幅图像的复合代数运算方程。

Matlab 图像处理工具箱提供了一些函数来进行图像的代数运算，见表 6.1。

表 6.1　Matlab 提供的图像代数运算函数

函 数	功　　能	函 数	功　　能
imadd	实现两个图像的加法运算	imcomplment	补足一幅图像
imsubtract	实现两个图像的减法运算	imabsdiff	计算两幅图像的差的绝对值
immultiply	实现两个图像的乘法运算	imlincomb	计算两幅图像的线形组合
imdivide	实现两个图像的除法运算		

说明如下。

（1）代数运算的结果很容易超出数据类型允许的范围，有些运算（如除法）会产生分数结果，这个结果是不能用整数类型描述的，所以，在图像的代数运算中采用以下的截取规则：

① 超出数据类型允许的范围的值用数据类型的最大值代替；

② 分数的结果采用四舍五入的方法取整。

（2）在使用这些函数的时候，不用进行数据类型的转换，这些函数可以接受 uint8 和 unit16 数据类型，并返回相同格式的图像结果。

（3）可以使用＋、－、＊、/等基本算术运算符来进行图像的算术操作运算，但是不同于函数，必须在运算前将图像转换为适合进行算术运算的双精度数据类型。

（4）任何一个代数运算，都必须保证输入图像的大小相等，且数据类型一致。

1. 图像相加运算

利用图像的加法运算给图像的每一个像素加上一个常数可以使图像的亮度增加。

【例 6.12】用图像的加法运算使图像 flowers.tif 的亮度增加。

```
RGB=imread('flowers.tif');
RGB2=imadd(RGB,50);              %对图像的每个像素加上一个常数
subplot(1,2,1);
imshow(RGB);
title('原图像');
subplot(1,2,2);
imshow(RGB);
title('增加亮度的图像');
```

其显示结果如图 6.14 所示。

图像相加也可以用来将一幅图像的内容叠加到另外一幅图像上，以达到二次曝光的效果。

【例 6.13】将图像 rice.tif 和图像 cameraman.tif 叠加到一起。



图 6.14　用代数运算增强图像亮度的图像对比

```
I=imread('rice.tif');
subplot(1,3,1)
imshow(I);
title('图像一');
J = imread('cameraman.tif');
subplot(1,3,2);
imshow(J);
title('图像二');
K = imadd(I,J,'uint16');               %实现两幅图像的叠加
subplot(1,3,3);
imshow(K,[]);
title('叠加后的图像');
```

其显示结果如图 6.15 所示。

图像一　　图像二　　叠加后的图像

图 6.15　两幅图像的叠加

在许多应用中，要得到一静止场景的许多幅图像是可能的。如果这些图像被一加性随机噪声源所污染，则可通过对多幅图像求平均值来达到降噪的目的。在求平均值的过程中，图像的静止部分不会改变；而对每一幅图像，各不相同的噪声图案则累积得很慢，通过对这些图像求平均值，可以有效地降低随机噪声的影响。

【例 6.14】通过求平均值降噪。

说明：在这里通过 Matlab 的 imnoise 函数对图像人为加入噪声，然后对多幅加入噪声的图像求平均值，达到去噪的目的。

```
clear;                                   %图像代数运算有相同大小，所以先清除变量
I=imread('rice.tif');
subplot(2,2,1);
imshow(I);title('原始图像');
[m,n]=size(I);
J(m,n)=0;
J=double(J);
X=imnoise(I,'gaussian');Y=double(X);            %加入噪声
subplot(2,2,2);
imshow(X);title('加噪图像一');
J=J+Y/10;
X=imnoise(I,'gaussian');Y=double(X);
subplot(2,2,3);
imshow(X);title('加噪图像二');
J=J+Y/10;
for i=1:8                                      %循环运算，对噪声的图像取平均值
   X=imnoise(I,'gaussian')
   Y=double(X);
   J=J+Y/10;
end
subplot(2,2,4);
imshow(mat2gray(J));title('10 幅噪声图像平均结果');
```

其显示结果如图 6.16 所示。

2. 图像相减运算

图像相减可以用于取出一幅图像中不需要的加性图案，加性团可能是缓慢变化的背景阴影、周期性的噪声，或在图像上每个像素处均已知的附加污染等。减法也可以用于检测同一场景的两幅图像间的变化。例如，通过对一场景的序列图像的减法运算可检测运动。在计算用于确定物体边界位置的梯度时，也要用到图像减法运算。

【例 6.15】比较余弦压缩变换图像和原图像的差别。

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);                  %产生二维 DCT 矩阵
%计算二维 DCT,T 和 T 转置是 DCT 函数 P1 * x * P2 的参数
B = blkproc(I,[8 8],'P1 * x * P2',T,T');
```

```
%二值掩模,用来压缩 DCT 系数
mask = [1   1   1   1   0   0   0   0
        1   1   1   0   0   0   0   0
        1   1   0   0   0   0   0   0
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1. * x',mask);          %只保留 DCT 变换的 10 个
系数
I2 = blkproc(B2,[8 8],'P1 * x * P2',T',T);       %重构图像
subplot(1,2,1);
imshow(I);title('原始图像');
subplot(1,2,2);
imshow(I2);title('压缩图像');
N=I2-I;           %两个图像相减
figure;imshow(N); title('差别图像');
```

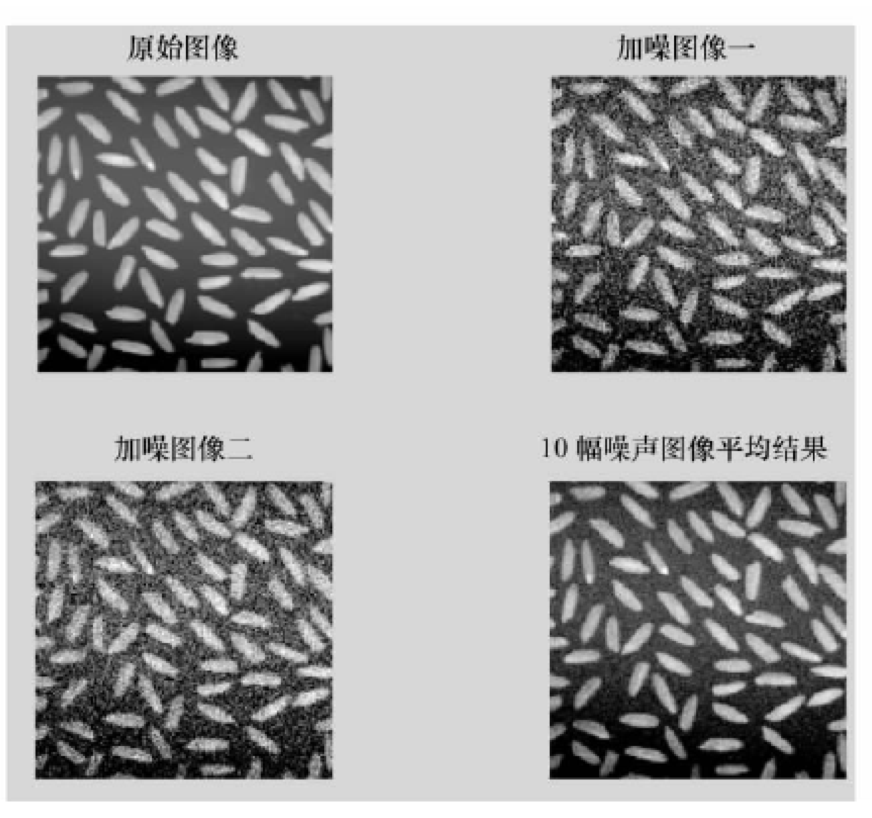


图 6.16　通过求平均值除噪的图像对比

其显示结果如图 6.17 所示。

【例 6.16】去掉图像 rice.tif 中的背景亮度图像。

说明:在以下代码中首先得到原始图像的背景图像亮度,然后从原始图像中减



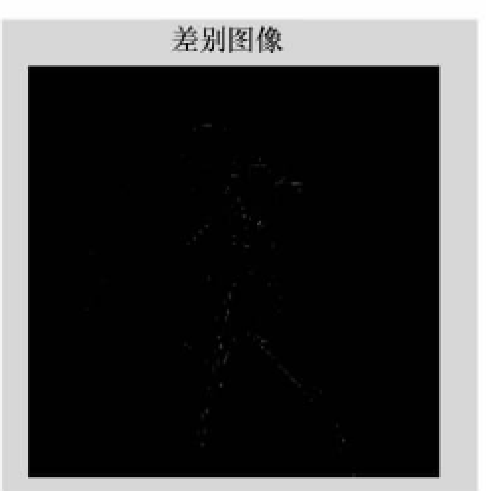


图 6.17　通过代数减法看压缩图像和原始图像差别

去背景图像亮度。

```
clear, close all
I = imread('rice.tif');
subplot(1,2,1)
imshow(I);
title('原始图像');
background = imopen(I,strel('disk',15));%得到背景图像
I2 = imsubtract(I,background);          %通过代数减法去掉图像中的背景图像
subplot(1,2,2);
imshow(I2);
title('去除背景的图像');
```

其显示结果如图 6.18 所示。

对图像进行减法代数运算时,有时会导致某些像素值的计算结果为负数,在这种情况下,imsubtract 函数会自动将这些像素值赋为 0。为了避免像素值之间的结果的差异,可以使用 imasbdiff 函数,函数 imabsdiff 计算的是两幅图像像素差值

的绝对值，这样计算的结果更加合理。

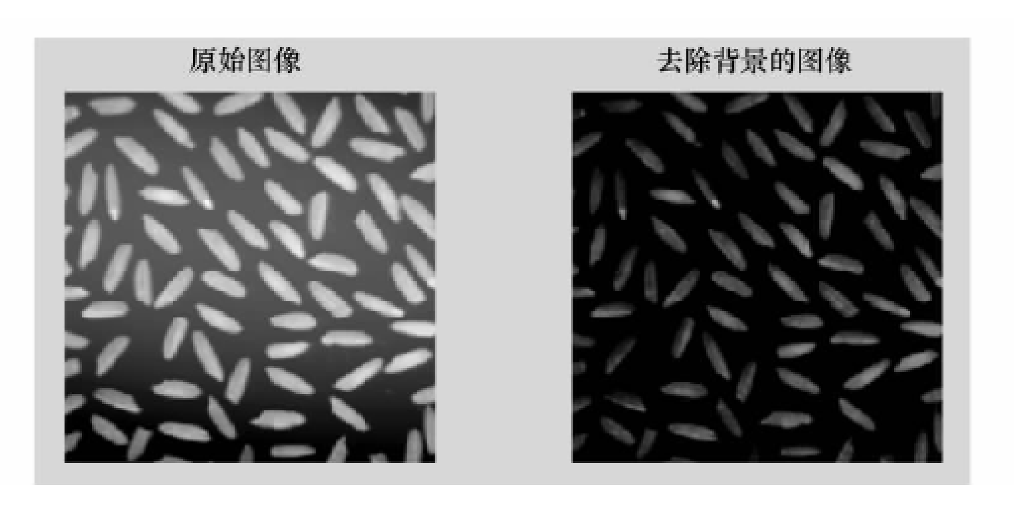


图 6.18　从图像中去掉背景图像亮度

6.3　空域滤波增强

6.3.1　基本原理

空域滤波是在图像空间中借助模板对图像进行邻域操作，输出图像每一个像素的取值都是根据模板对输入像素相应邻域内的像素值进行计算得到的。空域滤波基本上是让图像在频域空间内某个范围的分量受到抑制，同时保证其他分量不变，从而改变输出图像的频率分布，达到增强图像的目的。

一般情况下，像素的邻域比该像素大，即这个像素的邻域中除了本身以外还包括了其他许多像素。在这种情况下，输出像素 $g(x,y)$ 在 (x,y) 处的取值不仅取决于 $f(x,y)$ 在 (x,y) 处的值，而且取决于 $f(x,y)$ 在以 (x,y) 为中心的邻域内所有像素的值。如以 s 和 t 分别表示 $f(x,y)$ 和 $g(x,y)$ 在 (x,y) 处的灰度值，并且以 $n(s)$ 表示 $f(x,y)$ 在 (x,y) 邻域内像素的灰度值，则：

$$t = EH[s,n(s)]$$

为在邻域内实现增强操作，常可以利用模板与图像进行卷积。每个模板实际上是一个二维数组，其中各个元素的取值确定了模板的功能，这种模板操作被称为空域滤波。

模板运算的数学含义是一种卷积（或互相关）运算。主要步骤如下：

（1）将模板在图上游动，并将模板中心与图像的某个像素位置重合；

（2）将模板上的系数与模板下对应的像素相乘；

（3）将所有乘积相加；

（4）将和（模板的输出响应）赋给图中对应模板中心位置。

可以看出，这是一项非常耗时的运算。

【例 6.17】如图 6.19 所示，用模板 A 对图像矩阵 B 做模板运算。

2	2	2	2	3
3	1	1	(3)	2
2	2	3	4	4
3	3	4	5	6
5	5	6	6	6

(a)

1	1	1
1	5	1
1	1	1

(b)

图 6.19　图像矩阵和模板

(a)图像矩阵 B；(b)模板 A。

画圈的 3 为图像与模板重合的位置，在这个像素点模板运算的输出是：

$$2\times1+2\times1+3\times1+1\times1+3\times5+2\times1+3\times1+4\times1+4\times1=36$$

来看一下运算时间，每个像素完成一次模板操作要用 9 个乘法、8 个加法、1 个除法。对于一幅 $N\times N$（宽度×高度）的图像，就是 $9N^2$ 个乘法、$8N^2$ 个加法和 N^2 个除法，算法复杂度为 $o(N^2)$，这对于大图像来说，是非常可怕的。所以，一般常用的模板并不大，如 3×3，4×4。有很多专用的图像处理系统，用硬件来完成模板运算，大大提高了速度。另外，可以设法将二维模板运算转换成一维模板运算，对速度的提高也是非常可观的。

根据模板的特点，空域滤波一般分为线性滤波和非线性滤波两类。线性空域滤波常常是基于傅里叶分析的；非线性空域滤波器则直接对邻域进行操作。

按照空域滤波器的功能又可以将空域滤波器分为平滑滤波器和锐化滤波器两种。平滑滤波器可以用低通滤波实现，目的在于模糊图像（提取图像中的较大图像而消除小图像或将对象的小间断连接起来）或消除图像噪声。锐化滤波器可以使用高通滤波来实现的，目的在于强调图像被模糊的细节。

结合以上两种分类方法，将空间滤波增强方法分为以下四类：

（1）线性平滑滤波器（低通）；

（2）非线性平滑滤波器（低通）；

（3）线性锐化滤波器（高通）；

（4）非线性锐化滤波器（高通）。

6.3.2　平滑滤波器

1.线性平滑滤波器

线性平滑滤波器也称为均值滤波器。均值滤波器的所有系数都是正数，对 3×3模板来说，最简单的操作是所有系数都为 1，为了保持输出图像仍然在原来图像的灰度值范围内，模板与像素邻域的乘积都要除以 9。

【例 6.18】如图 6.20 所示，用 3×3 模板对图像进行线性平滑滤波。

2	2	2	2	3
3	1	1	(3)	2
2	2	3	4	4
3	3	4	5	6
5	5	6	6	6

(a)

1	1	1
1	5	1
1	1	1

(b)

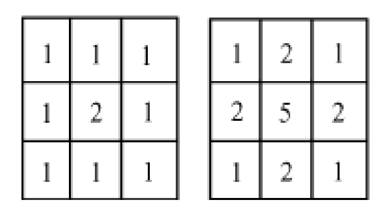
图 6.20　3×3 模板对图像进行线性平滑滤波

(a)图像矩阵;(b)3×3 模板。

画圈的 3 为图像与模板重合的位置,在这个像素点进行线性平滑滤波运算是:

$$(2\times1+2\times1+3\times1+1\times1+3\times5+2\times1+3\times1+4\times1+4\times1)/9=4$$

对上例的均值滤波器加以修改,就可以得到加权平均滤波器,如图 6.21 所示。



1	1	1
1	2	1
1	1	1

1	2	1
2	5	2
1	2	1

图 6.21　加权平均滤波器

Matlab 提供了 fspecial 函数来生成滤波时所用的模板,并提供 filter2 函数用指定的滤波器模板对图像进行运算。

函数:fspecial。

功能:创建一个指定的滤波器模板。

语法格式:

```
h = fspecial(type)
h = fspecial(type,parameters)
```

说明:参数 type 指定滤波器的种类;parameters 是与滤波器种类有关的具体参数。这两个参数的种类及含义见表 6.2。

表 6.2　Matlab 中预定义的滤波器种类

type	Parameters	说　明
average	hsize	均值滤波,如果邻域为方阵,则 hsize 为标量,否则由两元素向量 hsize 指定邻域的行数和列数
disk	radius	有(radius×2+1)个边的圆形均值滤波器
gaussian	hsize,sigma	标准偏差为 sigma,大小为 hsize 的高斯低通滤波器
laplacian	alpha	系数由 alpha(0.0~1.0)决定的二维拉普拉斯操作
log	hsize,sigma	标准偏差为 sigma,大小为 hsize 的高斯滤波旋转对称拉氏算子
motion	len,theta	按角度 theta 移动 len 个像素的运动滤波器
prewitt	无	近似计算垂直梯度的水平边缘强调算子

（续）

type	Parameters	说　　明
sobel	无	近似计算垂直梯度光滑效应的水平边缘强调算子
unsharp	alpha	根据 alpha 决定的拉氏算子创建的掩模滤波器

Matlab 工具箱中提供了一个函数 imnoise 来给图像增添噪声，下面将会用这个函数来给图像增加噪声，以方便示例。

函数：imnoise。

功能：给图像添加不同种类的噪声。

语法格式：

```
J = imnoise(I,type)
J = imnoise(I,type,parameters)
```

说明：

参数 type 指定噪声的种类；parameters 是与噪声种类有关的具体参数。这些参数可以具有的值见表 6.3。

表 6.3　噪声种类及参数说明

种　类	参数	说　明
gaussian	m,v	均值为 m,方差为 v 的高斯噪声
localvar	v	均值为 0,方差为 v 的高斯白噪声
passion	无	泊松噪声
salt pepper	无	椒盐噪声
speckle	v	均值为 0,方差为 v 的均匀分布随机噪声

【例 6.19】给图像加入椒盐噪声。

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);              %给图像加入椒盐噪声
subplot(1,2,1);
imshow(I);title('原始图像');
subplot(1,2,2)
imshow(J);title('加入椒盐噪声的图像');
```

其显示结果如图 6.22 所示。

【例 6.20】对一个图像进行不同大小模板的均值滤波，并比较结果。

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);  %对指定的图像加入椒盐噪声
subplot(2,2,1)
imshow(J);
```

```
title('噪声图像')；
K1=filter2(fspecial('average',3),J)/255；%进行 3×3 模板的均值滤波
K2=filter2(fspecial('average',5),J)/255；%进行 5×5 模板的均值滤波
K3=filter2(fspecial('average',7),J)/255；%进行 7×7 模板的均值滤波
subplot(2,2,2)
imshow(K1)；
title('3×3 模板均值滤波')；
subplot(2,2,3)
imshow(K2)；
title('5×5 模板均值滤波')；
subplot(2,2,4)
imshow(K3)；
title('7×7 模板均值滤波')；
```

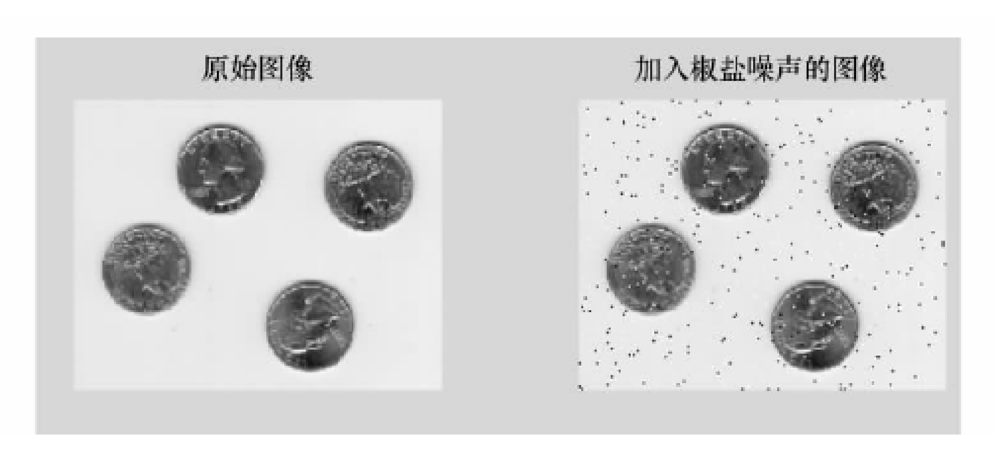


图 6.22 原始图像和加入噪声的图像

其显示结果如图 6.23 所示。

对以上用不同尺寸的滤波器模板进行滤波操作的图像进行比较，可以看到：随着所使用的滤波器尺寸增大的时候，消除噪声的效果得到了增强，但是，图像的细节锐化程度相应降低，图像变得模糊起来。

平滑模板的思想是通过一点和周围 8 个点的平均来去除突然变化的点，从而滤掉一定的噪声，其代价是图像有一定程度的模糊。前面提到的模板就是一种平滑模板，这种模板虽然考虑了邻域点的作用，但并没有考虑各点位置的影响，对于所有的 9 个点都一视同仁，所以平滑的效果并不理想。实际上离某点越近的点对该点的影响应该越大，为此，引入加权系数，将原来的模板改造成图 6.24 所示的新模板。

可以看出，距离越近的点，加权系数越大。新的模板其实也是一个常用的平滑模板，称为高斯(Gauss)模板。取这个名字的原因，是因为这个模板是通过采样二维高斯函数得到的。

1	2	1
2	4	1
1	2	1

图 6.24 高斯模板

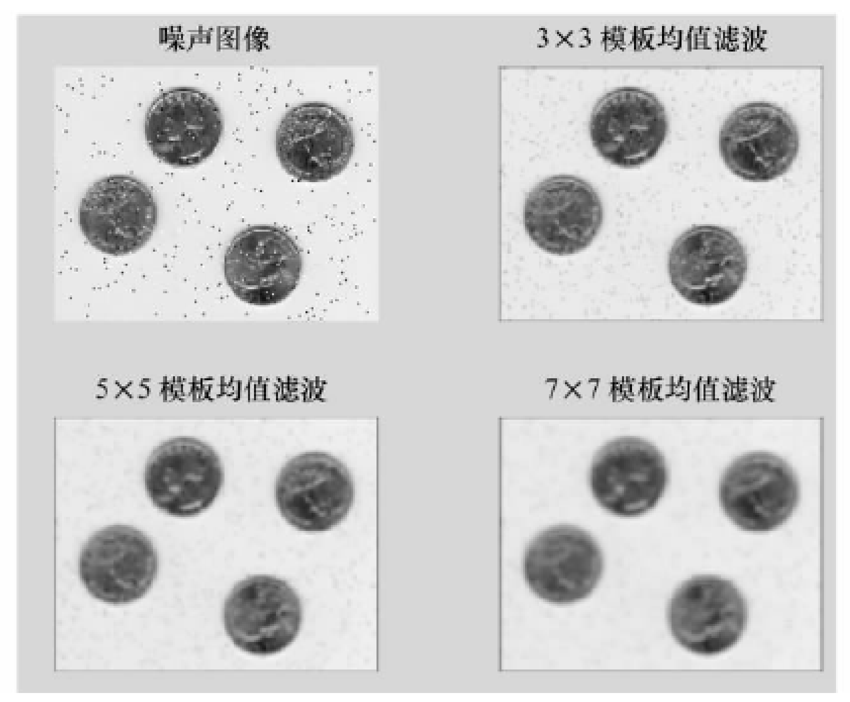


图 6.23　不同尺寸模板均值滤波的比较

2. 非线性平滑滤波器

中值滤波器是一种最常用的非线性平滑滤波器，其滤波原理与均值滤波器类似，但计算的不是加权求和，而是把邻域中的图像的像素按灰度级进行排序，然后选择改组的中间值作为输出像素值。

它们不同处在于：均值滤波器的输出是由平均值决定的，而中值滤波器的输出像素是由邻域图像的中间值决定的。所以中值滤波对极限像素值（与周围像素灰度值差别较大的像素）远不如平均值那么敏感，从而可以消除孤立的噪声点，又可以让图像产生较少的模糊。

Matlab 图像处理工具箱提供了 medfilt2 函数来实现中值滤波。

函数：medifilt2。

功能：实现对指定图像的中值滤波。

语法格式：

```
B = medfilt2(A,[m n])
B = medfilt2(A)
```

说明：

A 是输入图像；B 是中值滤波后输出的图像；[m n]指定滤波模板的大小，默认模板是 3×3 的。

【例 6.21】对一个图像实现不同模板的中值滤波，并比较结果。

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02); %对指定的图像加入椒盐噪声
subplot(2,2,1)
imshow(J);
```

```
title('噪声图像');
K1=medfilt2(J,[3 3]); %进行 3×3 模板的中值滤波
K1=medfilt2(J,[5 5]); %进行 5×5 模板的中值滤波
K1=medfilt2(J,[7 7]); %进行 7×7 模板的中值滤波
subplot(2,2,2)
imshow(K1);
title('3×3 模板中值滤波');
subplot(2,2,3)
imshow(K2);
title('5×5 模板中值滤波');
subplot(2,2,4)
imshow(K3);
title('7×7 模板中值滤波');
```

其显示结果如图 6.25 所示。

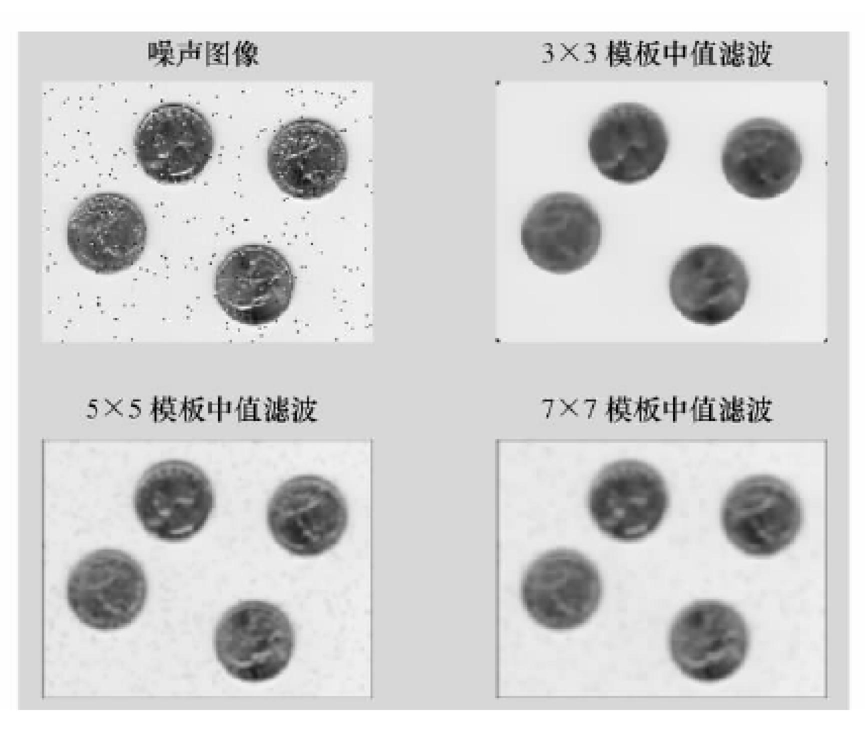


图 6.25 对图像进行不同模板的中值滤波

将图 6.25 中结果和均值滤波器进行比较,可以看到:中值滤波不像均值滤波器那样使图像的边界模糊,它在衰减噪声的同时,使图像的细节清楚。

6.3.3 锐化滤波器

图像平滑往往使图像中的边界、轮廓变得模糊,为了减少这类不利效果的影响,就需要利用图像锐化技术,使图像的边缘变得清晰。图像锐化处理的目的是使图像的边缘、轮廓线以及图像的细节变得清晰。经过平滑的图像变得模糊的根本原因是图像受到了平均或积分运算,因此对其进行逆运算(如微分运算)就可以使

图像变得清晰。从频域来考虑，图像模糊的实质是因为其高频分量被衰减，因此可以用高通滤波器来使图像清晰。

1.线性锐化滤波器

线性高通滤波器是最常用的线性锐化滤波器。这种滤波器必须满足滤波器的中心系数为正数，其他系数为负数。线性高通滤波器 3×3 模板的典型系数如图 6.26 所示。

-1	-1	-1
-1	8	-1
-1	-1	-1

图 6.26 线性高通滤波器 3×3 模板

事实上这是拉普拉斯算子，所有系数的和为 0。当这样的模板放在图像中灰度值是常数或变化很小的区域时，其输出为 0 或很小。有时会导致输出图像的灰度值为负数，而图像处理中一般仅考虑正灰度值，所以在这种情况下还要再进行灰度变换，使像素的灰度值保持在正整数范围内。

【例 6.22】对图像 saturn.tif 进行线性高通滤波。

```
I=imread('saturn.tif');
h=fspecial('laplacian'); %得到用于滤波的滤波器
I2=filter2(h,I);
subplot(1,2,1);
imshow(I);title('原始图像');
subplot(1,2,2);
imshow(I2);title('滤波后图像');
```

其显示结果如图 6.27 所示。

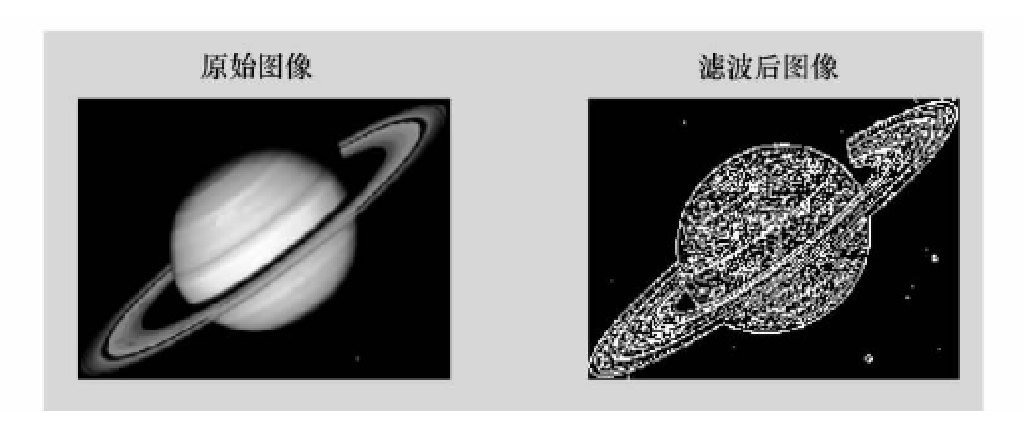


图 6.27 线性锐化滤波图像

2.非线性锐化滤波器

邻域平均可以模糊图像，因为平均对应积分，所以可以利用微分来锐化图像。非线性锐化滤波器就是应用微分对图像进行处理，其中最常用的就是利

用梯度，即图像沿某个方向上的灰度变化率。对于一个连续函数 $f(x,y)$，梯度定义如下：

$$\text{grad}[f(x,y)] = \left[\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right] \stackrel{def}{=} \Delta f \tag{6.2}$$

梯度是一个向量，需要用两个模板分别沿 x 和 y 方向计算。梯度的模（以 2 为模，对应欧氏距离）为：

$$|\Delta f| = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{\frac{1}{2}} \tag{6.3}$$

$$|\Delta f| = [(\Delta_x f)^2 + (\Delta_y f)^2]^{\frac{1}{2}} \tag{6.4}$$

其中：

$$\Delta_x = \frac{\Delta f}{\Delta x} = f(x+1,y) - f(x,y) \tag{6.5}$$

$$\Delta_y = \frac{\Delta f}{\Delta y} = f(x,y+1) - f(x,y) \tag{6.6}$$

常用的空域非线性锐化滤波微分算子有 sobel 算子、prewitt 算子、log 算子（高斯—拉普拉斯算子）等。

【例 6.23】用 sobel 算子、prewitt 算子、log 算子对图像滤波。

```
I=imread('rice.tif');
subplot(2,2,1);
imshow(I);title('原始图像');
h1=fspecial('sobel');
I1=filter2(h1,I);
subplot(2,2,2);
imshow(I1);title('sobel 算子滤波');
h1=fspecial('prewitt');
I1=filter2(h1,I);
subplot(2,2,3);
imshow(I1);title('prewitt 算子滤波');
h1=fspecial('log');
I1=filter2(h1,I);
subplot(2,2,4);
imshow(I1);title('log 算子滤波');
```

其显示结果如图 6.28 所示。

以上代码实现了对图像的非线性锐化滤波。

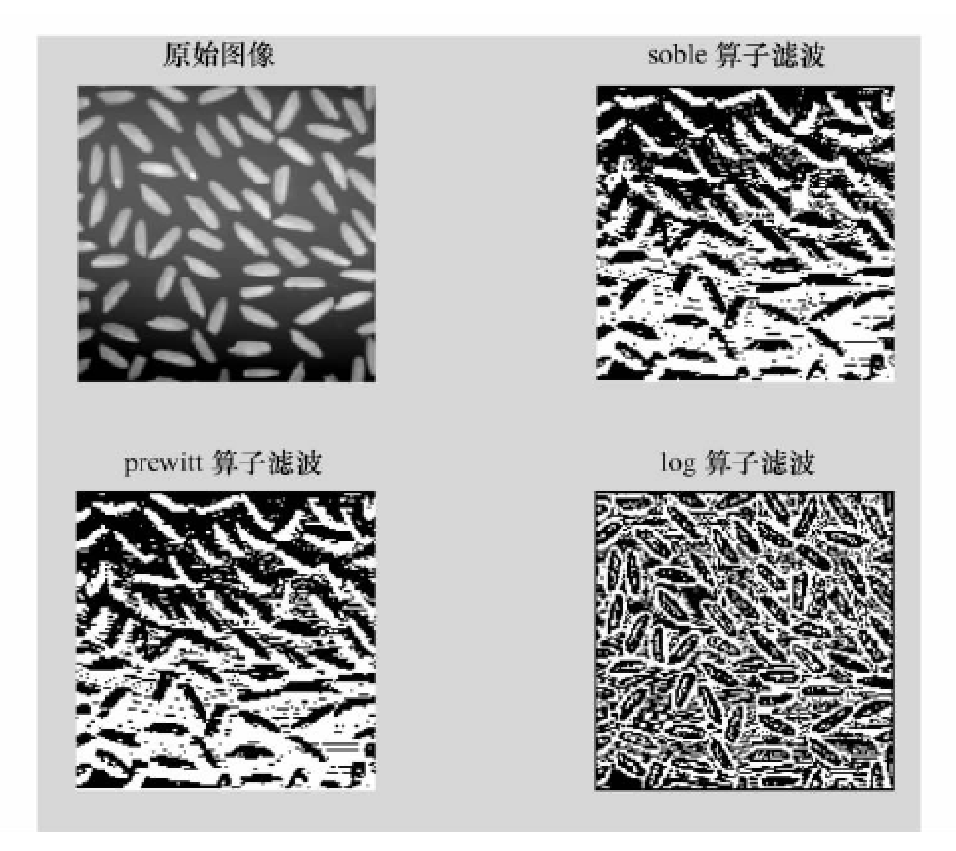


图 6.28　非线性锐化滤波

6.4　频域增强

频域增强是利用图像变换方法将原来的图像空间中的图像以某种形式转换到其他空间中，然后利用该空间的特有性质方便地进行图像处理，最后再转换回原来的图像空间中，从而得到处理后的图像。

频域增强的主要步骤是：

(1) 选择变换方法，将输入图像变换到频域空间；

(2) 在频域空间中，根据处理目的设计一个转移函数，并进行处理；

(3) 将所得结果用反变换得到增强图像。

常见的频域增强方法有低通滤波和高通滤波。

6.4.1　低通滤波

信号或图像的能量大部分集中在幅度谱的低频和中频段是很常见的；而在较高频段，感兴趣的信息常被噪声所淹没。因此，一个能降低高频成分幅度的滤波器就能减弱噪声的看得见的影响。

在傅里叶变换域，变换系数反映了某些图像特征。如频谱的直流分量对应于图像的平均亮度，噪声对应于频率较高的区域，图像实体位于频率较低的区域等。频域具有的这些内在特性常被用于图像增强，如构造一个低通滤波器，使低频分量

顺利通过而有效地阻止高频分量，即可滤除频域噪声，再经反变换来取得平滑图像。

由卷积定理，低通滤波数学表达式是：

$$G(u,v) = F(u,v)H(u,v) \tag{6.7}$$

式中：$F(u,v)$ 为含有噪声的原图像的傅里叶变换域；$H(u,v)$ 为传递函数（又称转移函数）；$G(u,v)$ 为经低通滤波后输出图像的傅里叶变换。

这里假定噪声和信号成分在频率上可分离，且噪声表现为高额成分。H 滤波滤去了高频成分，而低频信息基本无损失地通过。

选择合适的传递函数 $H(u,v)$对频率域低通滤波关系重大。常用的频域低通滤波器如下。

（1）理想圆形低通滤波器（Ideal Circular LowPass Filter，ICLPF）。它是一个在傅里叶平面上半径为 D_0 的圆形滤波器（见图 6.29），其传递函数为：

$$H(u,v) = \begin{cases} 1 & D(u,v) \leqslant D_0 \\ 0 & D(u,v) \geqslant D_0 \end{cases} \tag{6.8}$$

式中：D_0 为截止频率，大于 0；$D(u,v)$ 是点 (u,v) 到傅里叶频率域原点的距离。从理论上说，半径 D_0 内的频率分量无损通过，而圆外的频率分量会被滤除。在 D_0 适当的情况下，ICLPF 不失为简单易行的平滑工具。若滤除的高频分量中含有大量的边缘信息，将会发生图像边缘模糊现象。

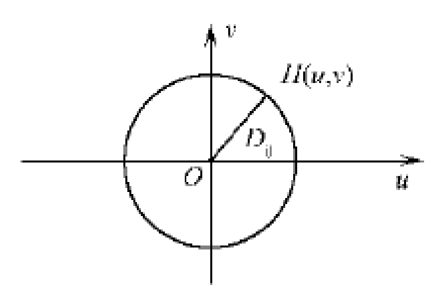


图 6.29 ICLPF 函数

（2）巴特沃斯低通滤波器（Butterworth LowPass Filter，BLPF）。该滤波器的转移函数由下式决定：

$$H(u,v) = \frac{1}{1 + (\sqrt{2} - 1)\left[\frac{D(u,v)}{D_0}\right]^{2n}} \tag{6.9}$$

函数曲线呈连续性衰减，而不像 ICLPF 曲线那样有陡峭的截止区。因此，BLPF 滤波后，图像边缘的模糊程度会大大减轻（曲线的“尾部”含有许多与边界相关的高频分量）。

（3）指数低通滤波器（Exponential Lowpass Filter，ELPF）。该滤波器的传递数为：

$$H(u,v) = e^{-\frac{[D(u,v)/D_0]^n}{\sqrt{2}}} \tag{6.10}$$

【例 6.24】对图像 eight.tif 加入椒盐噪声后，实现 Butterworth 低通滤波。

```
clear;
I1=imread('eight.tif');
subplot(2,2,1);
```

```
imshow(I1);title('原始图像');
I2=imnoise(I1,'salt & pepper');                          %加入椒盐噪声
subplot(2,2,2);
imshow(I2);title('噪声图像');
f=double(I2);
g=fft2(f)                                                %傅里叶变换
g=fftshift(g)                                            %转换数据矩阵
[N1,N2]=size(g);
n=2;
d0=50;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1                                               %对频域中循环滤波
        for j=2:N2
              d=sqrt((i-n1)^2+(j-n2)^2);
              %计算 Butterworth 低通滤波转换函数
              h=1/(1+0.414*(d/d0)^(2*n));
              result1(i,j)=h*g(i,j);
              if(g(i,j)>50)                              %进行理想低通滤波
                result2(i,j)=0;
              else
                result2(i,j)=g(i,j);
              end
        end
end
result1=ifftshift(result1);                              %进行反变换
result2=ifftshift(result2);                              %进行反变换
X2=ifft2(result1);
X3=uint8(real(X2));
subplot(2,2,3)
imshow(X3);
title('Btterworth 滤波图像');
X4=ifft2(result2);
X5=uint8(real(X4));
subplot(2,2,4)
imshow(X5);
title('理想低通滤波图像');
```

其显示结果如图 6.30 所示。

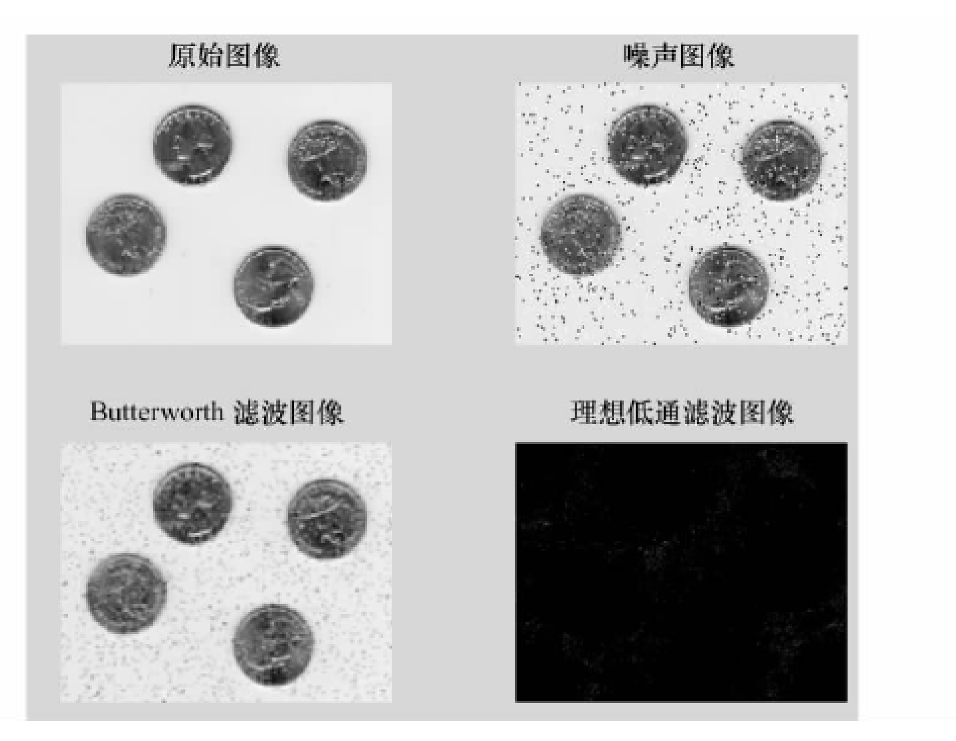


图 6.30　低通滤波图像

小波变换将一幅图像分解为大小、位置和方向都不同的分量。在做逆变换之前可以改变小波变换域中某些系数的大小，这样就能够有选择地放大所感兴趣的分量而减小不需要的分量。下面是一增强低频成分、衰减高频成分的实例。

【例 6.25】利用二维小波分解，实现对图像 woman 的增强处理。

分析：由于图像经二维小波分解后，图像的轮廓主要体现在低频部分，而细节则体现在高频部分，因此，可以通过对低频分解系数进行增强和对高频分解系数进行衰减处理，达到图像增强的作用。具体处理过程见如下程序。

```
load woman;
%画出原始图像
subplot(2,2,1);image(X);colormap(map);
title('原始图像');
axis square;
%下面进行图像的增强处理
%用小波函数 sym4 对 X 进行二层小波分解
[c,s]=wavedec2(X,2,'sym4');
sizec=size(c);
%对分解系数进行处理，通过处理，突出轮廓部分，弱化细节部分
for i=1:sizec(2)
   if(c(i)>350)
         c(i)=2*c(i);
   else
```

```
            c(i)=0.5*c(i);
        end
    end
    %下面对处理后的系数进行重构
        xx=waverec2(c,s,'sym4');
    %画出重构后的图像
        subplot(2,2,2);image(xx);
        title('增强图像')
```

其显示结果如图 6.31 所示。

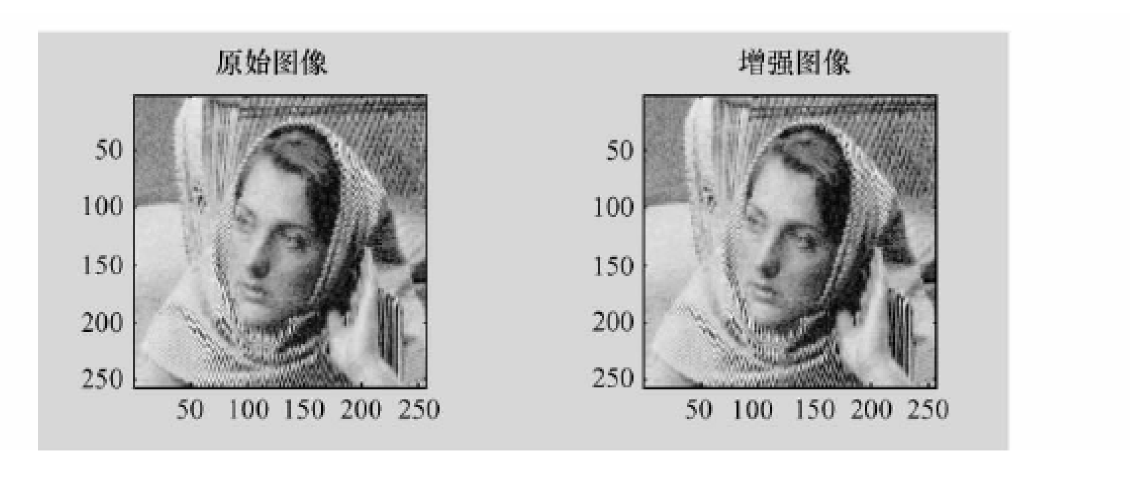


图 6.31 小波变换的低通滤波

6.4.2 高通滤波

由于图像中的细节部分与其频率的高频分量相对应,所以高通滤波可以对图像进行锐化处理。高通滤波与低通滤波的作用相反,它使高频分量顺利通过,而使低频分量受到削弱。频率域内常用的高通滤波器有四种,即理想高通滤波器、巴特沃斯高通滤波器、指数高通滤波器和梯形高通滤波器。参照前面的低通滤波器,不难分析高通滤波器的工作原理,这里仅列出这四种高通滤波器的转移函数 $H(u,v)$。

(1) 理想高通滤波器(Ideal HighPass Filter,IHPF)。是使特定频率区域的高频分量通过并保持不变,而其他频率区域的分量全部被抑制的滤波器。其转移函数为:

$$H(u,v)=\begin{cases}0 & D(u,v)\leqslant D_0\\ 1 & D(u,v)>D_0\end{cases}\tag{6.11}$$

(2) 巴特沃斯高通滤器(Butterworth HighPass Filter,BHPF)。该滤波器的转移函数为:

$$H(u,v) = \frac{1}{1 + (\sqrt{2} - 1)\left[\dfrac{D_0}{D(u,v)}\right]^{2n}} \tag{6.12}$$

（3）指数高通滤波器（Exponential HighPass Filter，EHPF）。该滤波器的转移函数为：

$$H(u,v) = \mathrm{e}^{-\frac{[D_0/D(u,v)]^n}{\sqrt{2}}} \tag{6.13}$$

式中：n 决定了指数函数的衰减率。当 $D(u,v) = D_0$，$n = 1$ 时，$H(u,v) = \mathrm{e}^{-1}$ 。

（4）梯形高通滤波器（Trapezoidal HighPass Filter ，THPF）。该滤波器的转移函数为：

$$H(u,v) = \begin{cases} 0 & D(u,v) < D_1 \\ \dfrac{D(u,v) - D_1}{D_0 - D_1} & D_1 \leqslant D(u,v) \leqslant D_0 \\ 1 & D(u,v) > D_0 \end{cases} \tag{6.14}$$

这是滤波特性介于理想滤波器和完全平滑滤波器之间的一种高通滤波器。

图像经过高通滤波处理后，会丢失许多低频信息，所以图像的平滑区基本上会消失。所以，可以采用高频加强滤波来弥补。高频加强滤波就是在设计滤波传递函数时，加上一个大于 0 小于 1 的常数 c，即：

$$H'(u,v) = H(u,v) + c \tag{6.15}$$

这种滤波被称为高频加强滤波，应用这种滤波，可以取得比一般高通滤波效果好的图像。

【例 6.26】对图像 eight.tif 实现 Butterworth 高通滤波。

```
clear;
I1=imread('eight.tif');
subplot(2,2,1);
imshow(I1);title('原始图像');
I2=imnoise(I1,'salt & pepper');                %加入椒盐噪声
subplot(2,2,2);
imshow(I2);title('噪声图像');
f=double(I2);
g=fft2(f)                                      %傅里叶变换
g=fftshift(g)                                  %转换数据矩阵
[N1,N2]=size(g);
n=2;
d0=50;
n1=fix(N1/2);
n2=fix(N2/2);
```

```
for i=1:N1                                              %对频域中循环滤波
        for j=2:N2
            d=sqrt((i-n1)^2+(j-n2)^2);    %进行 Butterworth 高通滤波
            if d==0
                h=0;
            else
                h=1/(1+(d0/d)^(2*n));
            end
            result1(i,j)=h*g(i,j);
            if(g(i,j)<50)                       %进行理想高通滤波
                result2(i,j)=0;
            else
                result2(i,j)=g(i,j);
            end
        end
end
result1=ifftshift(result1);                             %进行反变换
result2=ifftshift(result2);                             %进行反变换
X2=ifft2(result1);
X3=uint8(real(X2));
subplot(2,2,3)
imshow(X3);title('Butterworth 滤波图像');
X4=ifft2(result2);
X5=uint8(real(X4));
subplot(2,2,4)
imshow(X5);title('理想高通滤波图像');
```

其显示结果如图 6.32 所示。

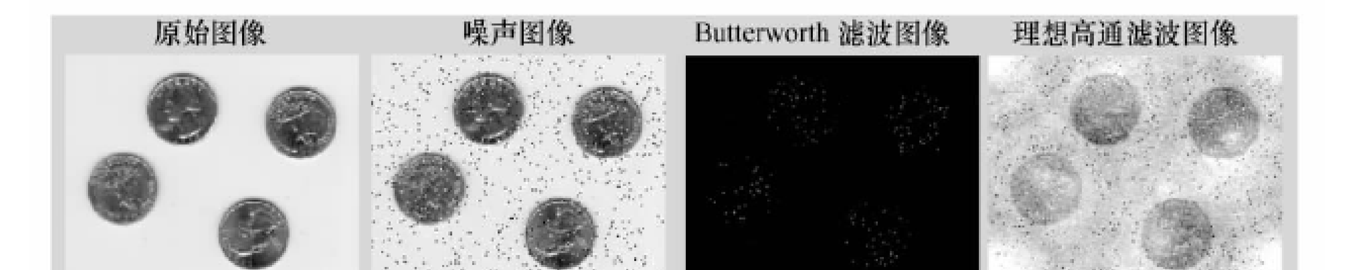


图 6.32　高通滤波图像

高通滤波器的滤波效果可以用原始图像减去低通滤波图像后得到。另外，也可以将原始图像乘以一个放大系数，然后再减去低通滤波图像后得到一幅高频率增强图像。

第 7 章　边缘提取和图像分割

图像理解是图像处理的一个重要分支，它要求计算机对图像进行分析和描述并对图像分析和理解。为了有效地进行图像理解，经常要求把给定图像进行分割或者将分割的图像区域用更加简单明确的数值、符号或图形表示出来。在这些方面经常要用到图像的边缘提取和图像分割技术。

7.1　边缘检测

数字图像的边缘检测是图像分割、目标区域的识别、区域形状提取等图像分析领域中十分重要的基础。边缘检测技术在数字图像处理中非常重要，因为边缘是图像中所要提取的目标和背景的分界线，只有提取出了边缘才能将背景和目标区分开来。

两个具有不同灰度值的相邻区域之间总存在边缘，边缘是灰度值不连续的结果。这种不连续性通常可以利用求导数的方法方便地检测到。一般常用一阶导数和二阶导数来检测边缘。

边缘检测的基本思想首先是利用边缘增强算子，突出图像中的局部边缘，然后定义像素的“边缘强度”，通过设值阈值的方法提取边缘点集。但是由于噪声和图像模糊的原因，检测到的边界可能会有间断的情况发生。所以边缘检测包含两个内容：

(1) 用边缘算子提取边缘点集；

(2) 在边缘点集中去除某些边缘点，填充一些边缘点，再将得到的边缘点集连接为线。

常用的检测算子有微分算子、log 算子和 canny 算子。

在 Matlab 图像处理工具箱中，提供了 edge 函数利用以上算子来检测灰度图像的边缘。

函数：edge。

功能：利用各种算子来做边缘检测。

语法格式：

```
BW=edge(I,method)
BW=edge(I,method,thresh)
```

```
BW = edge(I,method,thresh,direction)
```

说明：

I 是输入图像。method 是使用算子的类型，"sobel"是其默认值，表示用导数的 sobel 近似值检测边缘，那些梯度最大点返回边缘；为"prewitt"时表示用导数的 prewitt 近似值检测边缘，那些梯度最大点返回边缘；为"roberts"时表示用导数的 roberts 近似值检测边缘，那些梯度最大点返回边缘；为"log"时表示使用高斯滤波器的拉普拉斯运算对 I 进行滤波，通过寻找 0 相交检测边缘；为"zerocross"时表示用指定的滤波器对 I 进行滤波后，通过寻找 0 相交检测边缘；为"canny"时表示用拉普拉斯算子检测边缘。Thresh 是指定的阈值，所有不强于 thresh 的边都被忽略。Direction 是对于"sobel"和"prewitt"方法指定方向。Direction 是字符串，为"horizontal"表示水平方向，为"vertical" 表示垂直方向，为"both"表示两个方向（默认值）。BW 是返回的二进制图像。在二进制图像中，数值 1 代表找到的边缘，数值 0 代表其他像素。

edge 函数提供的最有效的边缘检测方法是 canny 方法。该方法的优点在于，使用两种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘和强边缘相连时，才将弱边缘包含在输出图像中。因此，这种方法不容易被噪声"填充"，更容易检测出真正的弱边缘。

7.1.1 微分算子法

微分算子具有突出灰度变化的作用，对图像运用微分算子，灰度变化较大的点处算得的值较高，因此这些微分值可作为相应的边界强度，所以可以通过对这些微分值设置阈值，提取边界的点集。

一阶导数是最简单的微分算子。已经在点 $f(x,y)$ 处，梯度 $\mathrm{grad}(f(x,y))$ 的幅度为：

$$\mathrm{grad}(f(x,y)) = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{\frac{1}{2}} \tag{7.1}$$

为了进一步简化计算，可以取：

$$\mathrm{grad}(f(x,y)) = |\Delta_x f| + |\Delta_y f| \tag{7.2}$$

或者

$$\mathrm{grad}(f(x,y)) = \max(|\Delta_x f| + |\Delta_y f|) \tag{7.3}$$

其中：

$$\Delta_x f = f(x,y) - f(x+1,y) \tag{7.4}$$

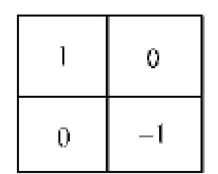
$$\Delta_y f = f(x,y) - f(x,y+1) \tag{7.5}$$

它们分别求出了灰度在 x 方向和 y 方向上的变化率，但是要对每一个像素进行以上的运算，运算量较大，所以在实际中采用小型模板利用卷积来做近似计算，

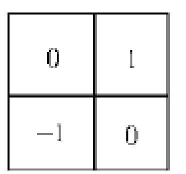
对 x 方向和 y 方向分别使用一个模板。现在经常使用的有 roberts 算子、sobel 算子和 prewitt 算子。

1. roberts 算子

roberts 算子是一种最简单的算子，是一种利用局部差分算子寻找边缘的算子，对具有陡峭的低噪声的图像效果最好。其模板如图 7.1 所示。



1	0
0	−1



0	1
−1	0

图 7.1　roberts 算子

2. sobel 算子

sobel 算子是滤波算子的形式，用于提取边缘，如图 7.2 所示。图像中的每个点都用这两个模板做卷积，第一个模板对垂直边缘影响最大；第二个模板对水平边缘影响最大。两个卷积的最大值作为该点的输出，运算结果是一幅边缘幅度图像。

1	2	1
0	0	0
−1	−2	−1

1	0	−1
2	0	−2
1	0	1

图 7.2　sobel 算子

3. prewitt 算子

与 sobel 算子相同，如图 7.3 所示。图像中的每个点都用这两个模板做卷积分，并且取最大值作为输出，结果是一幅边缘幅度的图像。

1	−1	1
1	−1	0
1	−1	−1

1	1	1
0	0	0
−1	−1	−1

图 7.3　prewitt 算子

【例 7.1】调用 sobel 算子、roberts 算子和 prewitt 算子检测图像 rice. tif 的边缘。

说明：在这里用 edge 函数调用不同的算子模板提取边缘。

```
I = imread('rice.tif');
BW1 = edge(I,'sobel');                %用 sobel 算子检测边缘
BW2 = edge(I,'roberts');              %用 roberts 算子检测边缘
BW3=edge(I, 'prewitt');               %用 prewitt 算子检测边缘
subplot(2,2,1);
imshow(I);title('原图像');
```

```
subplot(2,2,2);
imshow(BW1);title('sobel 算子提取结果');
subplot(2,2,3);
imshow(BW2);title('roberts 算子提取结果');
subplot(2,2,4);
imshow(BW3);title('prewitt 算子提取结果');
```

其显示结果如图 7.4 所示。

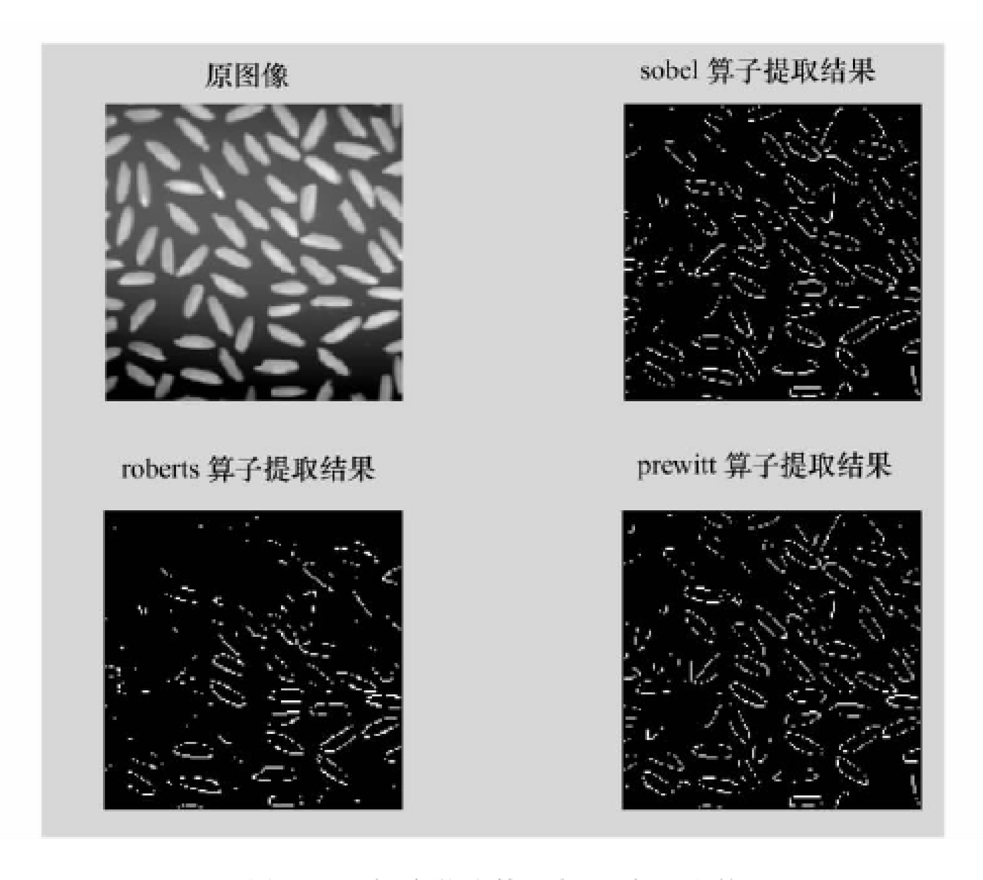


图 7.4　各种微分算子提取边界比较

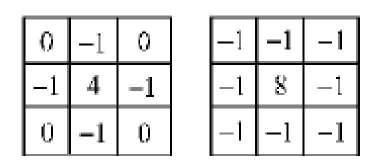
从提取图像可以看到，边缘的检测并不是十分准确。在对图像进行边缘检测时，由于灰度的变换并不是十分陡峭，或者由于噪声的干扰，会出现提取出错误的情况。所以如果在处理前（滤波）或处理后（连接）做一些工作，可以更加精确地检测边缘。

7.1.2　拉普拉斯—高斯算子法

拉普拉斯—高斯算子（Laplacian of Gaussian 算子，通常缩写为 log 算子）是一种二阶微分算子，将在边缘处产生一个陡峭的零交叉。log 算子是一个线性的、移不变的算子，它通过寻找图像灰度值中二阶微分是 0 的点来检测边缘点。对一个连续函数 $f(x,y)$，其在 (x,y) 处的 log 算子定义如下：

$$\Delta^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{7.6}$$

在图像边缘检测中，为了运算方便，函数的 log 算子也是借助模板来实现的。常用的模板如图 7.5 所示。



0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

图 7.5　拉普拉斯—高斯算子

拉普拉斯—高斯模板有一个基本要求：模板中心的系数为正，其余相邻系数为负，所有系数的和应该为零。

容易看出拉普拉斯—高斯模板的含义：先将自身与周围的 8 个像素相减，表示自身与周围像素的差别，再将这个差别加上自身作为新像素的灰度。可见，如果一片暗区出现了一个亮点，那么锐化处理的结果是这个亮点变得更亮，增加了图像的噪声。因为图像中的边缘就是那些灰度发生跳变的区域，所以锐化模板在边缘检测中很有用。

在 log 算子的检测过程中，首先要用高斯低通滤波器将图像进行预先平滑；然后利用 log 算子找到图像中的陡峭边缘；最后用 0 灰度值进行二值化，产生闭合的、连通的轮廓，消除所有内部点。

【例 7.2】用 log 算子检测图像 rice.tif 的边缘。

```
I=imread('rice.tif');
BW1 = edge(I,'log');                    %产生 log 算子
subplot(1,2,1);imshow(I);title('原图像');
subplot(1,2,2);imshow(BW1); title('log 算子检测图像');
```

其显示结果如图 7.6 所示。

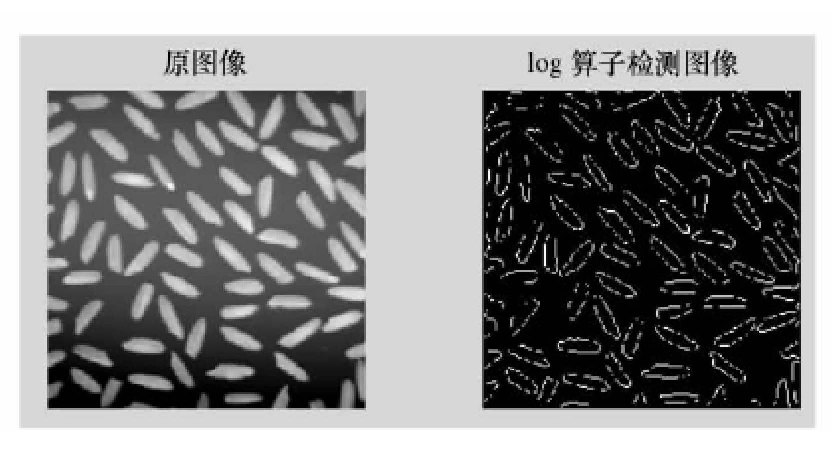


图 7.6　log 算子检测边缘

7.1.3　canny 法

canny 算子是一种比较新的边缘检测算子，具有很好的边缘检测性能，得到了越来越广泛的应用。canny 算子边缘检测的算法是寻找图像梯度的局部极大值，

梯度是用高斯函数的一阶微分来计算的。在 canny 法中，通过两个阈值来分别检测强边缘和弱边缘；当且仅当弱边缘与强边缘连接时，弱边缘才被在输出。canny 法不容易受噪声的干扰，能够在噪声和边缘检测之间取得较好的平衡，能够检测到真正的弱边缘。

【例 7.3】使用 canny 法检测图像 rice. tif 的边缘。

```
I=imread('rice.tif');
BW1 = edge(I,'canny');            %用 canny 法进行检测
subplot(1,2,1);imshow(I);title('原图像');
subplot(1,2,2);imshow(BW1); title('canny 算子检测图像');
```

其显示结果如图 7.7 所示。

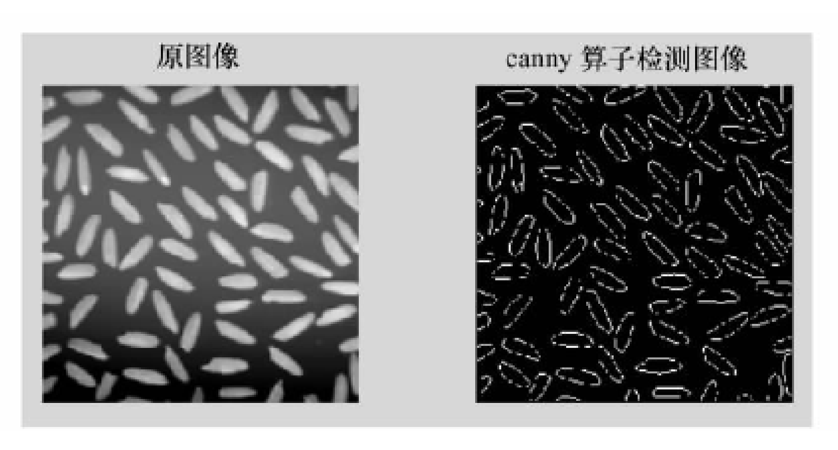


图 7.7　canny 法检测边缘

7.1.4　各种边缘检测算子的效果比较

【例 7.4】分别采用 roberts 算子、sobel 算子、prewitt 算子、canny 算子和 log 算子来检测图像 bloodl. tif 的边缘并比较。

```
I=imread('blood1.tif');
B1=edge(I,'roberts');
B2=edge(I,'sobel');
B3=edge(I,'prewitt');
B4=edge(I,'canny');
B5=edge(I,'log');
subplot(2,3,1);
imshow(I);title('原始图像');
subplot(2,3,2);
imshow(B1);title('roberts 算子检测');
subplot(2,3,3);
imshow(B2);title('sobel 算子检测');
subplot(2,3,4);
imshow(B3);title('prewitt 算子检测');
```

```
subplot(2,3,5);
imshow(B4);title('canny 算子检测');
subplot(2,3,6);
imshow(B5);title('log 算子检测');
```

其显示结果如图 7.8 所示。

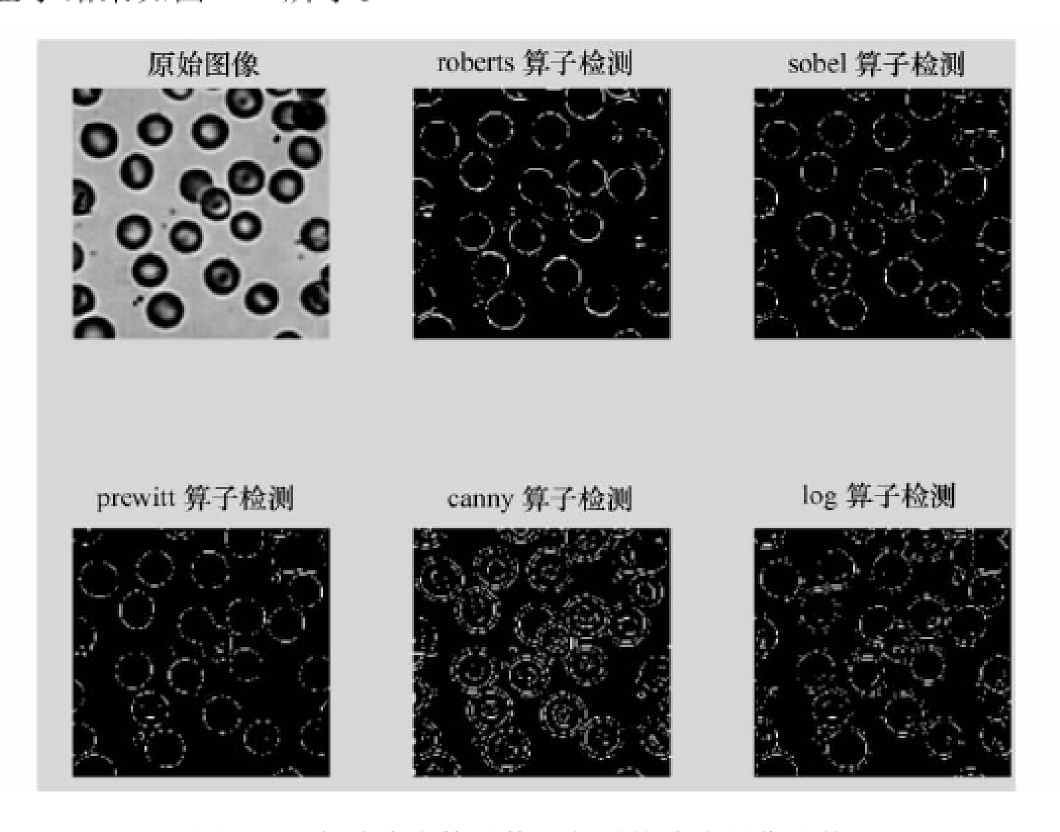


图 7.8　各种边缘检测算子得到的边缘图像比较

在对图像 blood1.tif 的分析中可以看到，使用 roberts 算子检测的效果比较好。但是，并不是每种图像都用 roberts 算子的效果好。各种算子适合使用的情况见表 7.1。

表 7.1　各种算子检测边缘情况表

算子	最佳情况
roberts	对具有陡峭的低噪声的图像效果较好。但是利用 roberts 算子提取边缘的结果是边缘比较粗，因此边缘定位不是很准确
sobel	对灰度渐变和噪声较多的图像值处理效果较好。sobel 算子对边缘定位比较准确
prewitt	对灰度渐变和噪声较多的图像值处理效果较好
log	经常出现双像素边界，并且该检测方法对噪声比较敏感，所以，很少用 log 算子检测边缘，而是用它来判断边缘像素是位于图像的明区还是暗区
canny	此方法不容易受噪声的干扰，能够检测到真正的弱边缘。在 edge 函数中，最有效的边缘检测方法是 canny 方法。该方法的优点在于，使用两种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘和强边缘相连时，才将弱边缘包含在输出图像中。因此，这种方法不容易被噪声“填充”，更容易检测出真正的弱边缘

一般来说,使用 canny 算子的边缘提取效果优于其他算子。

7.2 直线提取

利用微分算子对图像进行检测所得的边界常常会发生断裂现象,在这种情况下,需要用直线提取的方法把直线连接起来。另外在某些情况下,需要把直线单独提取出来。所以直线提取在图像处理中有重要的意义。

7.2.1 Hough 变换法

假设需要从图像的 n 个点中确定哪些点位于同一条直线上,那么可以将其看成是根据已知直线上的若干点来检测直线的问题。解决这一问题的一个直接方法就是先确定所有由任意两点决定的直线,然后从中找出接近具体直线的点集,这样做大约需要 $n^2 + n^3$ 次运算才能完成,实际上是很难满足这样大的计算量要求的。利用 Hough 变换就可以用较少的计算量来解决这一问题。

Hough 变换是利用图像全局特征将边缘像素连接起来形成封闭边界的一种连接方法,它把直线上的点的坐标转到过点的直线的系数域,巧妙地利用了共线和直线相交的关系。Hough 变换用来在图像中查找直线。它原理如图 7.9 所示。

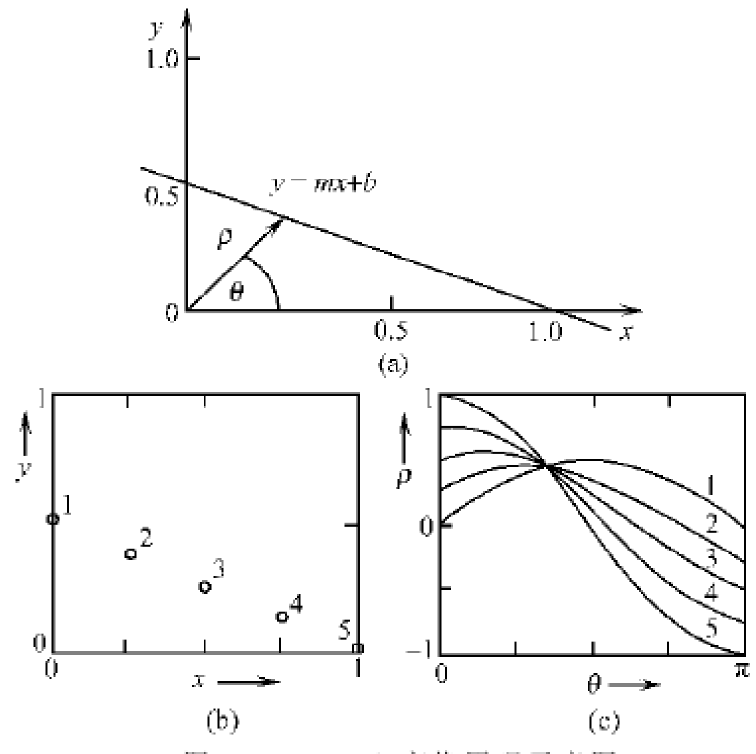


图 7.9 Hough 变换原理示意图

(a)一条直线的坐标表示;(b)x、y 平面;(c)ρ、θ 平面。

直线 $y = mx + b$ 可以用极坐标表示为:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{7.7}$$

其中 (ρ,θ) 定义了一个从原点到线上最近点的向量,这个向量与该直线垂直。

考虑一个以参数 ρ 和 θ 定义的二维空间，x、y 平面的任一直线对应了该空间的一个点。因此，x、y 平面的任一直线的 Hough 变换是 ρ、θ 空间中的一个点。

设点 (x_1, y_1) 是 x、y 平面的一个特定的点。过该点的直线可以有很多，每一条都对应了 ρ、θ 空间中的一个点，这些点，必须满足以 x_1 和 y_1 作为常量时的等式。因此在参数空间中与 x、y 空间中所有这些直线对应的点的轨迹是一条正弦型曲线，而 x、y 平面上的任一点对应了 ρ、θ 空间的一条正弦曲线。

如果有一组位于由参数 ρ_0 和 θ_0 决定的直线上的边缘点，则每个边缘点对应了 ρ、θ 空间的一条正弦曲线。所有这些曲线必交于点 (ρ_0, θ_0)，因为这是它们共享的一条直线的参数。为了找出这些点所构成的直线段，可以建立一个在 ρ、θ 空间的二维直方图。对每个边缘点(从小到大)，将给所有与该点的 Hough 变换(正弦曲线)对应的 ρ、θ 空间的直方图方格一个增量。当对所有边缘点施行完这种操作后，包含 (ρ_0, θ_0) 的方格将具有局部最大值。然后对 (ρ_0, θ_0) 空间的直方图进行局部最大值搜索，便可以获得边界线段的参数。

Hough 变换可能将不属于直线的点也连接到直接上，即产生所谓的过连接现象。在实际应用中，经常将图像划分为小块，然后对各个小块利用 Hough 变换提取直线，最后再将各段直线连接起来。

7.2.2 相位编组法

前面对边缘提取的方法采用的思路是：认为边缘是发生在灰度发生突变的地方，所以利用了梯度的幅度信息；当在某一个地方，图像的梯度幅度信息超过了预先设定的阈值，则认为这是边缘的存在点。但是边缘并不只是在灰度发生突变的地方存在，比如，在某个区域，灰度沿某个方向缓慢变化，这也是一种边缘，只是边缘比较粗。当在上述的情况下，利用前面的边缘检测方法不能取得很好的效果。

相位编组法采用了另外一种方法：它认为边缘不只是发生在梯度幅值较大的地方，即灰度发生突变的地方。因此，它引入了另一个判断参数：梯度的方向信息。相位编组法认为：当相邻点的梯度方向相同或者相近，则这个区域可能存在边缘。

相位编组法的计算方法如下：

(1) 计算图像上各个像素的梯度方向，计算方法为：

$$\alpha = \arctan\left[\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right]$$

(2) 根据图像上各点的梯度方向将这些点分为 8 个区域；

(3) 提取出图像上的边缘点；

(4) 从边缘点出发，将梯度方向在一个区域且相互间邻近的点划分在一个编组区(这可以通过从边缘点出发在其 8 个邻域内搜索方向一致的点来实现)；

(5) 用各相位编组区的点的灰度拟合灰度表面；

(6) 利用直线编组区的点的坐标和灰度表面求出直线的特性。

由于相位编组法可能受到噪声的影响,使各直线发生断裂,所以,还需要进行直线连接操作。

7.3 基于灰度分割

图像分割是一个将一幅数字图像划分为不交叠的、连通的像素集的过程,其中一个对应于背景,其他的则对应于图像中的各个物体。图像分割的一个难点在于:在划分前,不一定能够确定图像区域的数目。

7.3.1 灰度门限法

灰度门限分割是一种常见的直接检测区域的分割方法,它对物体与背景有较强对比的景物的分割特别有用。它计算简单,而且总能用封闭且连通的边界定义不交叠的区域。

利用灰度门限进行图像分割是基于一定的假设的,一般假设图像由具有单峰灰度分布的背景和目标组成,背景或目标内部的相邻像素间的灰度值是高度相关的,但是在目标和背景交界处的灰度值有很大的差别。如果图像满足这个条件,那么其灰度直方图基本上可以看成是由目标和背景两个单峰直方图组成的。如果这两个单峰直方图中单峰的情况接近并且均值相差较远、均方差较小,那么混合直方图应该是双峰的,这一类图像用灰度门限法就可以很好地进行分割。

当使用灰度门限法进行图像分割时,所有灰度值大于或等于某门限值的像素都被判属于物体,所有灰度值小于该门限值的像素被排除在物体之外。于是,边界就成为这样一些内部点的集合,这些点都至少有一个邻点不属于该物体。如果感兴趣的物体在其内部具有均匀一致的灰度值并分布在一个具有另一个灰度值的均匀背景上,使用灰度门限法效果就很好。如果物体同背景的差别在于某些性质而不是灰度值(如纹理等),那么,可以首先把那个性质转化为灰度,然后,利用灰度门限法分割待处理的图像。

7.3.2 灰度门限的确定

分割门限选择的准确性直接影响分割的精度及图像描述分析的正确性。门限选得太高,容易把大量的目标判定为背景,定得太低又会把大量的背景判为目标。因此,正确分割门限很重要。

以下介绍几种选取阈值的方法。

1. 全局阈值化方法

采用阈值确定边界的最简单做法是在整个图像中将灰度阈值的值设置为常

数。如果背景的灰度值在整个图像中可合理地看做恒定，而且所有物体与背景都具有几乎相同的对比度，那么，只要选择了正确的阈值，使用一个固定的全局阈值一般会有较好的效果。

一般是根据图像的直方图来选择阈值。如果图像的灰度直方图是一个双峰一谷直方图（其中一个峰值对应目标的中心灰度，另一个峰值对应背景的中心灰度），那么就选择两峰间的谷值作为阈值来分割目标和背景。

一方面由于直方图的参照性，另一方面由于直方图有可能受到噪声的干扰，所以实际上寻找阈值并不是一个简单的过程，需要设计一定的准则进行搜索。下面列出一些寻找全局化阈值的方法：

（1）通过搜索直方图的两个最大的局部最大值，假设它们的位置是 h_1 和 h_2，并且它们的距离大于 H（一个指定的距离），然后找在 h_1 和 h_2 间直方图的最小值 h_m，再用 h_m 作为分割的门限。

（2）把阈值设在相对于两峰的某个固定位置，如中间位置上。一般情况下，对这些参数的估计比对最少出现的灰度值即直方图的谷底的估计更可靠。

（3）当两峰一谷的情况比较显著的情况下，如果阈值对应于直方图的谷底，阈值从 T 增加到了 $T+\Delta T$ 只会引起面积略微减少。因此把阈值设在直方图的谷底，可以把阈值选择中的小错误对面积测量的影响降到最低。

如果图像或图像包含的物体区域面积不大且有噪声，那么，直方图本身就会有噪声。除了凹谷特别尖锐的情况外，噪声会使谷底的定位难以辨认。这个问题在一定程度上可以通过用卷积或曲线拟合过程对直方图进行平滑加以克服。如果两峰大小不一样，那么，平滑化可能会导致最小值的位置发生移动。但是，在平滑化程度适当的情况下，谷值还是容易定位并且也是相对稳定的。

2. 自适应阈值法

在许多情况下，背景的灰度值并不是常数，物体和背景的对比度在图像中也有变化。这时，一个在图像中某一区域效果良好的阈值在其他区域却可能效果很差。在这种情况下，把灰度阈值取成一个随图像中位置缓慢变化的函数值是适宜的。

7.4 四叉树分解

除了采用灰度门限的方法对图像进行分割，还有一些其他的分割方法，如还可以采用区域生长的方法，即从图像中的一些种子点开始，按照一定的一致性准则将邻近的像素包含进去产生区域。

7.4.1 四叉树分解原理及 Matlab 工具箱函数

四叉树分解是一种常用的分裂合并分割方法。分裂合并分割方法的基本思想

是从整幅图像开始不断分裂得到各个区域，最终将具有一致性的像素分到同一个小块。这里的一致性是指各个像素点的灰度值的接近程度要满足要求。一致性标准可以选择如下的形式：

(1)区域中灰度最大值与最小值的差小于阈值；

(2)两区域平均灰度差小于阈值；

(3)两区域的灰度分布函数差小于阈值；

(4)两区域参数统计特征结果相同。

在实际应用中，还可以根据应用来设计满足要求的一致性标准。

四叉树分解的具体过程是：将一块图像分成四块等大小的方块，判断每个块是否满足一致性标准，如果满足，则不再分解；如果不满足，再进行细分成四块，并且对细分得到的方块应用一致性准则检查，分解过程重复迭代下去，直到满足一致性准则。结果可能包含不同大小的块。

四叉树分解的示意图如图 7.10 所示，R 代表整个图像区域，可以将 R 连续地分裂为越来越小的 1/4 正方形子区域 R_i，并保证每一个子区域符合预先设定的一致性标准。

R_{11} R_{12} R_2
R_{13} R_{14}
R_3 R_4

图 7.10　四叉树分解示意图

但是对图像进行分解的同时，还需要对图像分裂的块进行合并。如果只对图像进行分裂而不合并，那么最后可能导致相邻的两个区域具有相同性质，但是却被划分为两个区域，所以在每一次分裂操作后还允许子区域按照一定的准则进行合并。

在 Matlab 图像处理工具箱中提供了 qtdecomp 函数来进行四叉树分解。这个函数首先将图像划分为相等大小的四块，然后对每一个块进行一致性标准检查。如果某个块符合一致性标准，那么就不对其进行进一步的分割；如果不符合一致性标准，则将该块继续分为四块，将测试标准应用于其他块。这个过程将会一直重复直至每一个块都符合这个标准。分解的结果可能会包含许多大小不同的块。

对一幅 64×64 的灰度图像进行四叉树分解，图像首先会被划分为四个 32×32 的块，然后对每一个块应用一致性标准检查，假设一致性标准如下：

```
max(block(:)-min(block(:))<=0.2
```

如果其中一个块符合一致性标准，则这个块将不会再进行分解，分解完成后还是一个 32×32 的块，然后再次对这些块应用相同的一致性标准。不符合一致性标准的块将被再次分解为四个 16×16 的块，然后再对这些块应用相同的一致性标

准。不符合标准的块将再次分解为四个 8×8 的块，依此类推，直至所有的块都通过测试，有些块可能只有 1×1 大小。

四叉树分解得到的块一般都是方块，也有少量的长方形块。当图像是方形，且宽度是 2 的整数次幂时，四叉树法最适用。

观察图 7.11 中四叉树分解的结果，可以看到其中有不同大小的方块，这些方块是在各次分解中得到的。

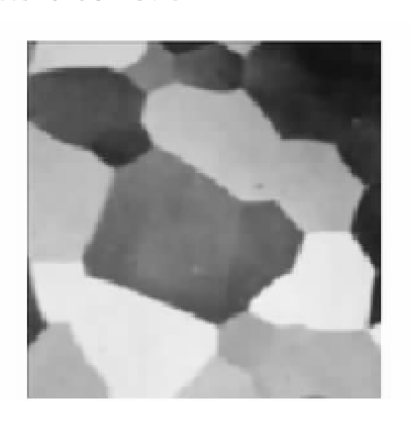
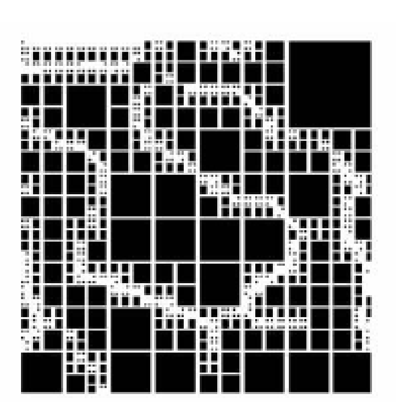



图 7.11 使用四叉树分解图像

7.4.2 应用四叉树分解

在 Matlab 图像处理工具箱中提供了以下函数来实现四叉树分解。

1. 函数 qtdecomp

功能：对指定图像进行四叉树分解。

语法格式：

```
S = qtdecomp(I)
S = qtdecomp(I,threshold)
S = qtdecomp(I,threshold,mindim)
S = qtdecomp(I,threshold,[mindim maxdim])
S = qtdecomp(I,fun)
S = qtdecomp(I,fun,P1,P2,...)
S =qtdecomp(I,threshold,[mindim,maxdim])
```

说明：

I 是输入图像矩阵；threshold 是一个可选参数，如果某个子区域中的最大像素值减去最小值大于 threshold，则分解块；[mindim，maxdim]也是可选参数，用来指定最终分解得到的子区域大小，如果子区域不在这个区域中，那么分解继续进行，否则返回；***S*** 是一个稀疏矩阵，其非 0 元素的位置对应于块的左上角坐标，每一个非 0 元素的数值代表块的大小。***S*** 的大小与 ***I*** 相同。

S=qtdecomp(I)对灰度图像矩阵 ***I*** 进行四叉树分解，返回四叉树结构的稀疏矩阵 ***S***。

S=qtdecomp(I,threshold)表示如果块中元素最大值减去最小值大于 threshold,则分解块,threshold 为 0~1 的值。

S=qtdecomp(I,threshold,mindim)表示如果块小于 mindim 就不再进行分解,无论其符合阈值条件与否。

S = qtdecomp(I,threshold,[mindim maxdim])表示如果块小于 mindim 或大于 maxdim 就不再进行分解。maxdim/mindim 必须为 2 的幂。

无论 ***I*** 是何种类型的,阈值都是一个指定在 0 到 1 间的数值。如果 ***I*** 是 uint8 类型的,那么指定的阈值将会乘以 255 作为真实的阈值使用;如果 ***I*** 是 unit16 类型的,那么使用的阈值将会乘以 65535。

【例 7.5】在工作台中输入图像矩阵,并以阈值 5 进行四叉树分解。

在工作台中输入如下代码:

```
I=[ 1    1    1    1    2    3    6    6
    1    1    2    1    4    5    6    8
    1    1    1    1    10   15   7    7
    1    1    1    1    20   25   7    7
    20   22   20   22   1    2    3    4
    20   22   22   20   5    6    7    8
    20   22   20   20   9    10   11   12
    22   22   20   20   13   14   15   16];
S = qtdecomp(I,5);      %进行四叉树分解,返回稀疏矩阵 S
full(S);                %将稀疏矩阵 S 转换为完整的矩阵
```

其结果显示在 ans 中:

```
ans =
    4    0    0    0    2    0    2    0
    0    0    0    0    0    0    0    0
    0    0    0    0    1    1    2    0
    0    0    0    0    1    1    0    0
    4    0    0    0    2    0    2    0
    0    0    0    0    0    0    0    0
    0    0    0    0    2    0    2    0
    0    0    0    0    0    0    0    0
```

实际上不用把 S 画出即可获得图像四叉树分解的图像块信息。

2. 函数 qtgetblk

功能:得到四叉树分解后子块像素及其位置信息。

语法格式:

```
[vals,r,c] = qtgetblk(I,S,dim)
[vals,idx] = qtgetblk(I,S,dim)
```

说明：

I 是四叉树分解的图像的返回图像的矩阵；***S*** 是由 qtdecomp 函数返回的稀疏矩阵；dim 是所想得到的维数；vals 是一个 dim×dim×*k* 维的数组 ，*k* 是四叉树分解中 dim×dim 维方块的个数，如果分解结果中没有指定尺寸的子块，则返回空矩阵；*r* 是存放行向量的坐标；*c* 是存放列向量的坐标；idx 是存放返回分解的子块中的左上角的线性索引。

数据类型要求：***I*** 可以是 uint8 型、uint16 型或 double 型；***S*** 是 sparse 型的。

数组 vals 中方块的顺序必须与 I 中方块的列顺序相对应。例如，如果 vals 是 4×4×2 的，vals(：，：，1)包含的值就来自于 I 中的第一个 4×4 方块，vals(：，：，2)中的值就用来替换 I 中的第二个 4×4 方块。

【例 7.6】对例 7.5 中的图像 I 进行四叉树分解，提取分解结果中大小为 4×4 的图像块的信息。

在工作台中输入如下代码：

```
I=[ 1    1    1    1    2    3    6    6
    1    1    2    1    4    5    6    8
    1    1    1    1    10   15   7    7
    1    1    1    1    20   25   7    7
    20   22   20   22   1    2    3    4
    20   22   22   20   5    6    7    8
    20   22   20   20   9    10   11   12
    22   22   20   20   13   14   15   16];
S = qtdecomp(I,5);
[vals,r,c] = qtgetblk(I,S,4)
```

得到如下结果：

```
vals(:,:,1) =

      1     1     1     1
      1     1     2     1
      1     1     1     1
      1     1     1     1

vals(:,:,2) =

     20    22    20    22
     20    22    22    20
     20    22    20    20
     22    22    20    20
```

3. 函数 qtsetblk

功能：将图像 I 的四叉树分解中的每个 dim×dim 维子块的值全部替换为 vals 中的 dim×dim 维的方块。

语法格式：

```
J=qtsetblk(I,S,dim,vals)
```

说明：

I 是四叉树分解的图像矩阵；***S*** 为由 qtdecomp 函数返回的稀疏矩阵，包含了四叉树结构；dim 是所进行替代子块的维数；vals 是一个 dim×dim×*k* 维的数组，*k* 是四叉树分解中 dim×dim 维方块的个数。

数据类型要求：***I*** 可以是 uint8 型、uint16 型或 double 型；***S*** 是 sparse 型的。

数组 vals 中方块的顺序必须与 I 中方块的列顺序相对应。例如，如果 vals 是 4×4×2 的，vals(:,:,1)包含的值就用来替换 I 中的第一个 4×4 方块，vals(:,:,2)包含的值就用来替换 ***I*** 中的第二个 4×4 方块。

【例 7.7】对 I 分解的结果调用 qtsetblk。

在工作台中输入如下代码：

```
I=[ 1    1    1    1    2    3    6    6
    1    1    2    1    4    5    6    8
    1    1    1    1    10   15   7    7
    1    1    1    1    20   25   7    7
    20   22   20   22   1    2    3    4
    20   22   22   20   5    6    7    8
    20   22   20   20   9    10   11   12
    22   22   20   20   13   14   15   16]
S = qtdecomp(I,5);
[vals,r,c] = qtgetblk(I,S,4) ;
newvals = cat(3,zeros(4),ones(4));
J = qtsetblk(I,S,4,newvals)
```

输出结果如下：

```
J=[ 0    0    0    0    2    3    6    6
    0    0    0    0    4    5    6    8
    0    0    0    0    10   15   7    7
    0    0    0    0    20   25   7    7
    1    1    1    1    1    2    3    4
    1    1    1    1    5    6    7    8
    1    1    1    1    9    10   11   12
    1    1    1    1    13   14   15   16 ]
```

【例 7.8】对图像 rice.tif 进行四叉树分解，并显示稀疏矩阵。

```
subplot(1,2,1);
imshow(I);
title('原始图像');
```

```
S= qtdecomp(I,0.2);                    %取阈值为 0.2,对图像进行四叉树分解
N=full(S);
subplot(1,2,2);
imshow(N);
title('分解的图像显示');
[vals,r,c]=qtgetblk(I,N,2);
[vals1,r,c]=qtgetblk(I,N,4);
[vals2,r,c]=qtgetblk(I,N,8);
[vals3,r,c]=qtgetblk(I,N,16);
[vals4,r,c]=qtgetblk(I,N,32);
[vals5,r,c]=qtgetblk(I,N,1);
size(vals);size(vals1);size(vals2);
size(vals3);size(vals4);size(vals5);
```

输出结果:

```
ans=1   1   872
ans=2   2   4166
ans=4   4   1540
ans=8   8   317
ans=16  16  12
ans =32 32  0
```

其显示结果如图 7.12 所示。

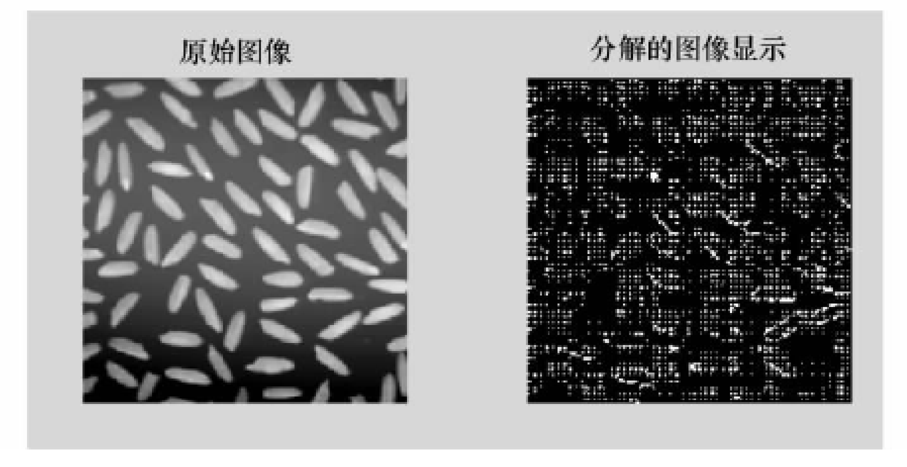


图 7.12　四叉树分解结构

第 8 章　二值形态学操作

8.1　二值形态学基本运算

8.1.1　数学形态学简介

数学形态学是一种应用于图像处理和模式识别领域的新方法，基本思想是用具有一定形态的结构元素去度量和提取图像中的对应形状，以达到对图像分析和识别的目的。用于描述二值形态学的语言是集合论，因此它可以用一个统一而强大的工具来处理图像处理中所遇到的问题。利用数学形态学对物体几何结构的分析过程就是主客体相互逼近的过程。利用数学形态学的几个基本概念和运算，将结构元灵活地组合分解，应用形态变换达到分析的目的。

数学形态学方法比其他空域或频域图像处理和分析方法具有一些明显的优势。例如，在图像恢复处理中，基于数学形态学的形状滤波器可借助于先验的几何特征信息，利用形态学算子有效地滤除噪声，又可以保留图像中的原有信息；另外，数学形态学算法易于用并行处理方法有效地实现，而且硬件实现容易；基于数学形态学的边缘信息提取处理优于基于微分运算的边缘提取算法，它不像微分算法对噪声那样敏感，同时，提取的边缘比较平滑，提取的图像骨架也比较连续，断点少。

集合论是数学形态学的基础，下面首先对集合论的一些基本概念作一总结性的概括介绍。

集合（集）：具有某种性质的确定的有区别的事物的全体（它本身也是一个事物）。常用大写字母如 $A,B\cdots$ 表示。如果某种事物不存在，就称这种事物的全体是空集。规定任何空集都只是同一个集，记为 ϕ。在以下的介绍中设 A、B、C 等均是 n 维欧氏空间 E^n 的集合。

元素：构成集合的每个事物。常用小写字母如 $a,b,\cdots$ 表示。任何事物都不是 ϕ 中的元素。

子集：当且仅当集合 A 的元素都属于集合 B 时，称 A 为 B 的子集。

并集：由 A 和 B 的所有元素组成的集合称为 A 和 B 的并集。

交集：由 A 和 B 的公共元素组成的集合称为 A 和 B 的交集。

补集：A 的补集记为 A^c，定义为

$$A^c = \{x \mid x \notin A\} \tag{8.1}$$

位移：A 用 $x=(x_1,x_2)$ 位移，记为$(A)x$，定义为

$$(A)_x = \{y \mid y = a + x, a \in A\} \tag{8.2}$$

映像：A 的映像（也称映射）记为 $\hat{A}$，定义为

$$\hat{A} = \{x \mid x = -a, a \in A\} \tag{8.3}$$

差集：两个集合 A 和 B 的差，记为 A、B，定义为

$$A、B = \{x \mid x \in A, x \notin B\} = A \cap B^c \tag{8.4}$$

8.1.2　数学形态学基本运算

1. 膨胀

A、B 为 Z 中的集合，ϕ 为空集，A 被 B 的膨胀记为 $A \oplus B$，$\oplus$ 为膨胀算子，膨胀的定义为：

$$A \oplus B = \{x \mid [\hat{B}_x \cap A] \neq \phi\} \tag{8.5}$$

式(8.5)表明的膨胀过程是 B 首先作关于原点的映射，然后平移 x，这里 A 与 B 映像的交集不为空。A 被 B 的膨胀过程也就是 $\hat{B}$ 的位移与 A 至少有一个非零元素相交时 B 的中心像素的位置的集合。

同在其他的形态处理中一样，集合 B 在膨胀操作中通常被称为结构元素。

图像膨胀的过程如图 8.1 所示。

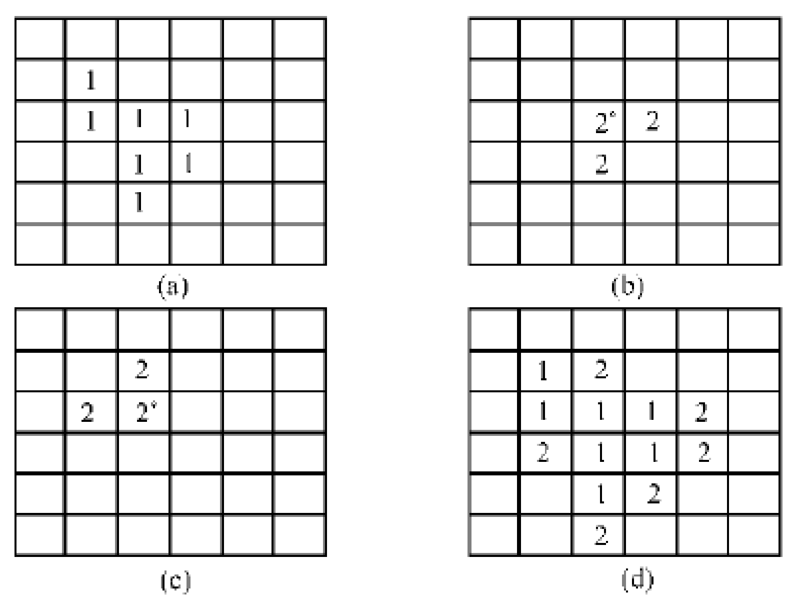


图 8.1　图像的膨胀过程

图 8.1(a)中为 1 的部分是集合 A，图 8.1(b)中为 2 的部分为集合 B，其中 2^* 表示像素的中心位置。图 8.1(c)是图 8.1(b)的映像，图 8.1(d)中标有 1 的像素为 A 中原来的位置，标有 2 的像素表示膨胀出来的部分，合起来就是膨胀的结果，也就是 A⊕B。

2. 腐蚀

A、B为Z中的集合，A被B腐蚀记为$A\ominus B$，其定义为$A\ominus B=\{x\mid (B)_x\subseteq A)\}$。也就是说$A$被$B$腐蚀的结果为所有使$B$被$x$平移后包含于$A$的点的集合。

膨胀和腐蚀是关于集合补和反转的对偶，即$(A\ominus B)^c = A^c \oplus B^c$。

图像腐蚀的过程如图8.2所示。

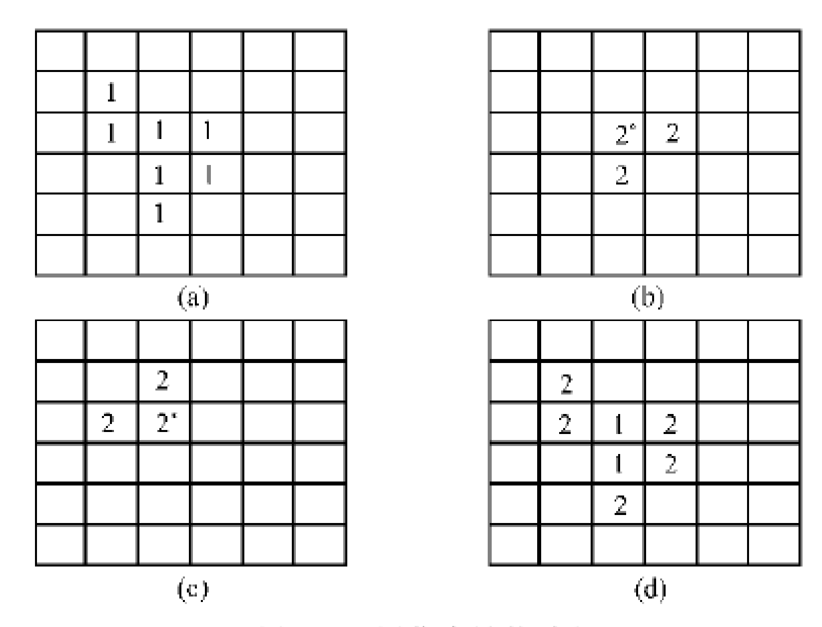


图 8.2　图像腐蚀的过程

图8.2(a)中为1的部分是集合A，图8.2(b)中为2的部分为集合B，其中2*表示像素的中心位置。图8.2(c)是图8.2(b)的映像，图8.2(d)中标有1的像素为A中原来像素保留下来的位置，标有2的像素表示A中被腐蚀掉的像素，标有1的像素是腐蚀的结果，也就是A⊖B。

3. 开启运算和闭合运算

开启运算就是先对图像进行腐蚀，然后对腐蚀的结果做膨胀操作。开启运算符为○开其运算定义为

$$A\bigcirc B = (A\ominus B)\oplus B \tag{8.6}$$

开启运算一般能平滑图像的轮廓，削弱狭窄的部分，去掉细的突出。

闭合运算是先对图像进行膨胀，然后对膨胀的结果做腐蚀运算。闭合运算符为·。闭合运算定义为

$$A\cdot B = (A\oplus B)\ominus B \tag{8.7}$$

闭合运算也是平滑图像的轮廓，与开启运算相反，它一般能融合窄的缺口和细长的弯口，去掉小洞，填补轮廓上的缝隙。

开启运算和闭合运算是关于集合补和反转的对偶，即$(A\cdot B)^c = A^c\bigcirc B^c$。

4. 膨胀和腐蚀的对偶性

膨胀和腐蚀这两种操作有着密切的关系：一种运算使用结构元素对图像的操作相当于另一种运算使用结构元素对图像背景的操作，用公式表示为：

$$(A \oplus B)^c = A^c \ominus B^c \tag{8.8}$$

$$A^c \oplus B^c = (A \ominus B)^c \tag{8.9}$$

图像膨胀和腐蚀的对偶性如图 8.3 所示。

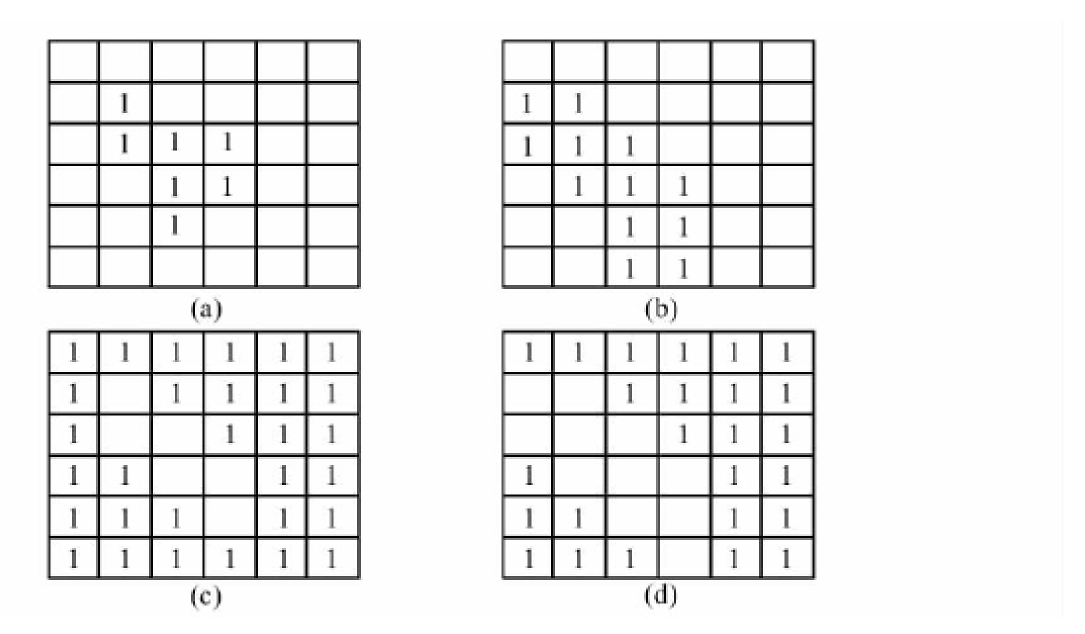


图 8.3　图像膨胀和腐蚀的对偶性

图 8.3(a)是原始图像，图 8.3(c)是背景图像，图 8.3(b)是原始图像的膨胀结果，图 8.3(d)是背景图像的腐蚀结果。从图中可以看到，对前景的膨胀结果和对背景的腐蚀结果相同。

8.1.3　图像形态学

将数学形态学推广至灰度形态学，与前面不同的是以下的讨论中将处理数字图像函数而不是集合。设 $f(x,y)$ 是输入图像，$b(x,y)$ 是结构元素，它可被看做是一个子图像函数。如果 $\mathbf{Z}$ 表示实整数集合，同时假设(x,y) 是来自 $\mathbf{Z}\times\mathbf{Z}$ 的整数，f 和 b 是坐标为(x,y) 像素灰度值的函数。如果灰度也是整数，则 Z 可由整数 R 所代替。

1. 膨胀

在 Matlab 图像处理工具箱中，膨胀一般是给图像中的对象边界添加像素，进行膨胀操作时，输出像素值是输入图像相应像素邻域内所有像素的最大值。在二进制图像中，如果任何一个像素值为 1，那么对应的输出像素值为 1。

二值图像的膨胀运算的过程如图 8.4 所示，因为结构元素定义的邻域中有一个元素数值为 1，所以对应的输出像素取值为 1。

用结构元素 b 对输入图像 f 进行灰度膨胀，记为 $f \oplus b$，其定义为：

$$(f \oplus b)_{(s,t)} = \max\{f(s-t,t-y)+b(x,y) \mid (s-x),(t-y) \in D_f \text{ 和} (x,y) \in D_b\} \tag{8.10}$$

其中 D_f 和 D_b 分别是 f 和 b 的定义减。

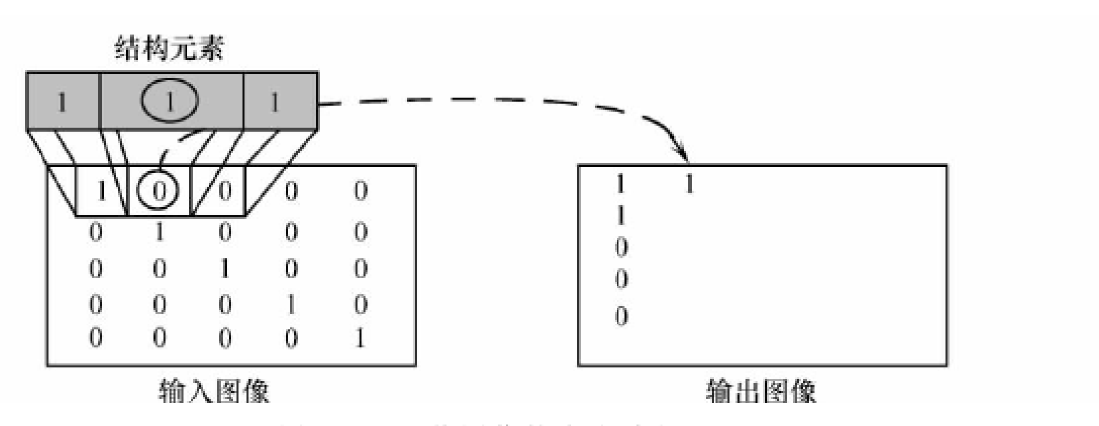


图 8.4　二值图像的膨胀过程

灰度图像的膨胀如图 8.5 所示。

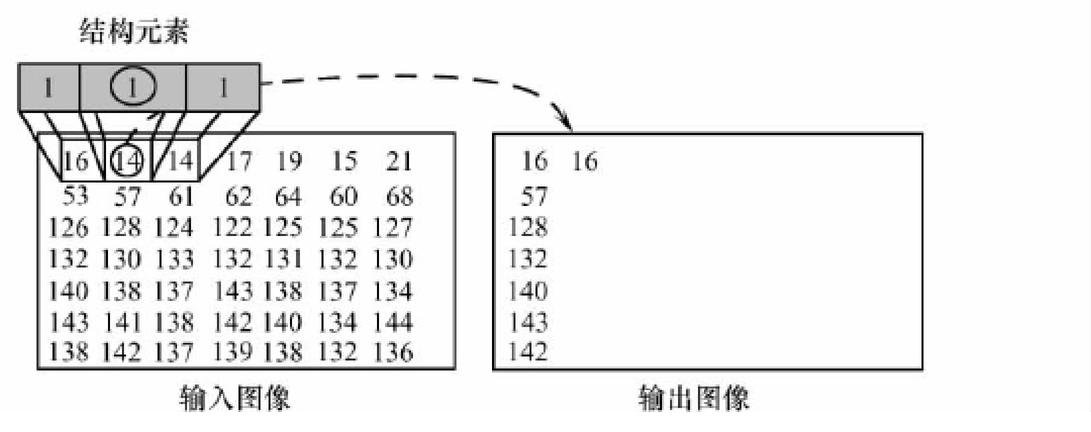


图 8.5　灰度图像的膨胀示意图

膨胀的计算是在由结构元素确定的邻域中选取 $f+b$ 的最大值，所以对灰度图像的膨胀操作有两个效果：

（1）如果结构元素的值都为正的，则输出图像会比输入图像亮。

（2）根据输入图像中暗细节的灰度值以及它们的形状相对于结构元素的关系，它们在膨胀中或被消减或被除掉，如图 8.6 所示。

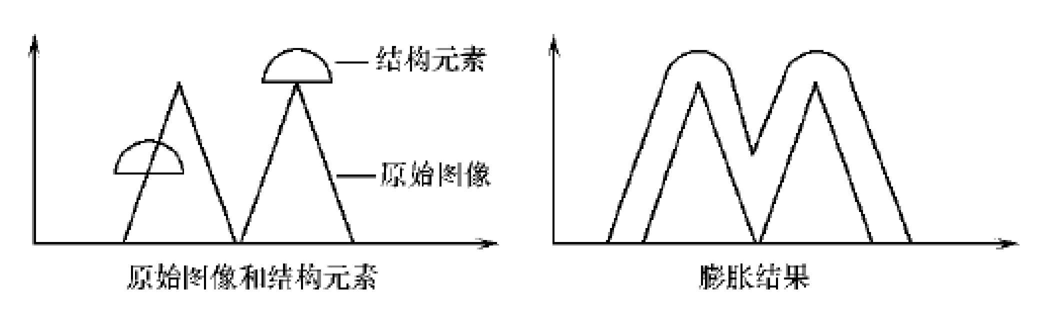


图 8.6　膨胀结构示意图

2. 腐蚀

在 Matlab 图像处理工具箱中，腐蚀则是删除图像边界元素，输出像素值是输入图像相应像素邻域内所有像素的最小值。在二进制图像中，如果任何一个像素值为 0，那么对应的输出像素值为 0。

用结构元素 b 对输入图像 f 进行灰度膨胀记为 $f \ominus b$，其定义为：

$$f \ominus b_{(s,t)} = \min\{f(s-t, t-y) + b(x,y) \mid (s-x),(t-y) \in D_f \text{ 和}(x,y) \in D_b\} \tag{8.11}$$

其中 D_f 和 D_b 分别是 f 和 b 的定义域。这里限制 $(s-x)$ 和 $(t-y)$ 在 f 的定义域之内，类似于在二值腐蚀定义中要求结构元素完全包括在被腐蚀集合中。

腐蚀的计算是在由结构元素确定的邻域中选取 $f-b$ 的最小值，所以对灰度图像的腐蚀操作有两个效果：

（1）如果结构元素的值都为正，则输出图像会比输入图像暗；

（2）如果输入图像中亮细节的尺寸比结构元素小，则其影响会被减弱，减弱的程度取决于这些亮细节周围的灰度值和结构元素的形状及幅值。

3．开启和闭合

由于膨胀和腐蚀并不是互为逆运算，所以可以将它们级联结合使用。开启就是先对图像进行腐蚀，然后对腐蚀的结果做膨胀运算。闭合是先对图像做膨胀运算，然后对膨胀运算的结果做腐蚀运算。这两种运算是数学形态学中的重要运算。

灰度数学形态学中关于开启和闭合的表达与它们在二值数学形态学中的对应运算是一致的。

开启消除了尺寸较小的亮细节，闭合消除了尺寸较小的暗细节。

4．结构元素

在通常情况下，形态学图像处理以在图像中移动一个结构元素并进行一种类似于卷积操作的方式进行。像卷积核一样，结构元素可以具有任意大小，也可以包含任意的 0 与 1 的组合。在每个像素位置，结构元素核与在它下面的二值图像之间进行一种特定的逻辑运算。逻辑运算的二进制结果存在输出图像中对应于该像素的位置上。产生的效果取决于结构元素的大小、内容以及逻辑运算的性质。

在 Matlab 中进行膨胀和腐蚀计算，主要就是利用结构元素。二维（平面）的结构元素是由一个数值为 0 或 1 的矩阵组成，通常比要处理的图像小得多。结构元素的原点指定了图像中需要处理的像素的范围，结构元素中数值为 1 的点决定了结构元素邻域中的像素在进行膨胀或腐蚀操作时是否需要参与计算。三维或非平面的结构元素使用 0 和 1 来定义结构元素在 x 和 y 平面上的范围，采用第三维来定义高度。

结构元素本身也是一个图像矩阵。对每个结构元素矩阵指定一个原点，在 Matlab 中称做中心像素（见图 8.7），它是结构元素参与形态学运算的参考点，通常表示用户期望的像素。在 Matlab 中，中心像素定义如下：

floor((size(SE)+1/2)

其中，SE 是结构要素矩阵。结构元素在 Matlab 中被定义为一个 STREL 对象，由于 Matlab 规定不能在表达式中直接使用对像本身的大小，所以必须使用 STREL 对像的 SE 属性来获得结构元素的邻域。

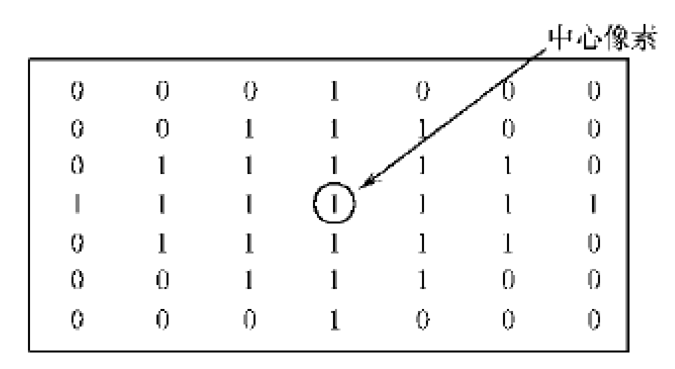


图 8.7　钻石形状的结构元素

可以使用 Matlab 图像处理工具箱中的 strel 函数来创建任意大小和形状的 STREL 对象。strel 函数支持许多种常用的形状，如线形（Line）、钻石形（Diamond）、圆盘形（Disk）和球形（Ball）等。

【例 8.1】创建一个平面钻石形结构元素。

```
se=strel('diamond',3)
```

得到以下的运算结果：

```
se= 0    0    0    1    0    0    0
    0    0    1    1    1    0    0
    0    1    1    1    1    1    0
    1    1    1    1    1    1    1
    0    1    1    1    1    1    0
    0    0    1    1    1    0    0
    0    0    0    1    0    0    0
```

为了提高执行效率，strel 函数可能会将结构元素拆为较小的块，这种技术通常称为结构元素分解。例如，使用一个 11×11 正方形结构元素的膨胀操作可以首先用 1×11 的结构元素进行膨胀，然后再使用 11×1 的结构元素进行膨胀，这样做在理论上会使执行速度提高 6.5 倍。圆盘形和球形结构元素的分解结果是近似的，其他形状的分解结果是精确的。如果希望观察分解得到的结构元素序列，可以调用函数 getsequence。getsequence 函数返回一个分解后的结构元素数组，例如，以下代码将获得一个钻石形结构元素分解所得到的结构元素的序列。

```
se=strel('diamond',3);
seq=getsequence(se);
```

得到以下结果：

```
seq(1)
    0    1    0
    1    1    1
    0    1    0
seq(2)
    0    1    0
```

```
        1    0    1
        0    1    0
    seq(3)
        0    1    0
        1    0    1
        0    1    0
```

结构元素中的原点都定义在对输入图像感兴趣的位置。对于图像边缘的像素，由结构元素定义的邻域将会有一部分位于图像边界外。为了处理边界像素，一般进行形态学运算的函数都会给超出图像、未指定数值的像素指定一个数值，这看起来好像是函数给图像填充了额外的行和列。膨胀和腐蚀操作的像素的填充值是不同的，对于二进制图像和灰度图像，膨胀和腐蚀操作使用的填充方法具体如下。

（1）膨胀：超出图像边界的像素值定义为该数据类型允许的最小值。对于二进制图像，这些像素值被设置为0；对于灰度图像，unit8类型的最小值也是0。

（2）腐蚀：超出图像边界的像素值定义为该数据类型允许的最大值。对于二进制图像，这些像素值被设置为1；对于灰度图像，unit8类型的最大值是255。

通过对膨胀操作使用最小值填充和对腐蚀操作使用最大值填充可以消除边界效应（输出图像靠近边界处的区域与图像其他部分不连续）。如果腐蚀操作使用最小值进行填充，那么图像腐蚀将会导致输出图像围绕一个黑色边框。

8.1.4 Matlab中二值形态学运算

1．膨胀运算

Matlab图像处理工具箱提供了imdilate函数进行图像膨胀。

函数：imdilate。

功能：对指定的图像作膨胀运算。

语法格式：

```
IM2 = imdilate(IM,SE)
IM2 = imdilate(IM,NHOOD)
IM2 = imdilate(IM,SE,PACKOPT)
IM2 = imdilate(...,PADOPT)
```

说明：

IM是输入图像；IM2是膨胀运算后的输出图像；SE是结构元素对象，SE可以是一个定义结构元素邻域的二进制矩阵或是由strel函数返回的对象；PACKOPT和PADOPT是可选参数，PADOPT参数用来影响输出图像的大小，PACKOPT参数用来说明输入图像是否为打包二进制图像。

【例8.2】对图像text.tif做膨胀操作，并进行对比。

```
bw = imread('text.tif');
%创建一个线形结构元素
se = strel('line',11,90);
%用线形结构元素来进行膨胀操作
bw2 = imdilate(bw,se);
subplot(1,2,1);
imshow(bw), title('原始图像');
subplot(1,2,2);
imshow(bw2), title('膨胀图像');
```

其显示结果如图 8.8 所示。



图 8.8　图像和膨胀图像的对比

2. 腐蚀运算

函数:imerode。

功能:对指定的图像作腐蚀操作。

语法格式:

```
IM2 = imerode(IM,SE)
IM2 = imerode(IM,NHOOD)
IM2 = imerode(IM,SE,PACKOPT,M)
IM2 = imerode(...,PADOPT)
```

说明:

IM 是输入图像;IM2 是膨胀运算后的输出图像;SE 是结构元素对像,SE 可以是一个定义结构元素邻域的二进制矩阵或是由 strel 函数返回的对象;PACKOPT、PADOPT 和 M 是可选参数,PADOPT 参数用来影响输出图像的大小,PACKOPT 参数用来说明输入图像是否为打包二进制图像,如果图像是打包二进制图像,那么 M 将指定原始图像的行数。

【例 8.3】对图像 text.tif 作腐蚀操作,并进行对比。

```
bw= imread('text.tif');
se= strel('line',11,90);
```

```
bw2= imerode(bw,se);
subplot(1,2,1);
imshow(bw), title('原始图像');
subplot(1,2,2);
imshow(bw2), title('腐蚀图像');
```

其显示结果如图 8.9 所示。

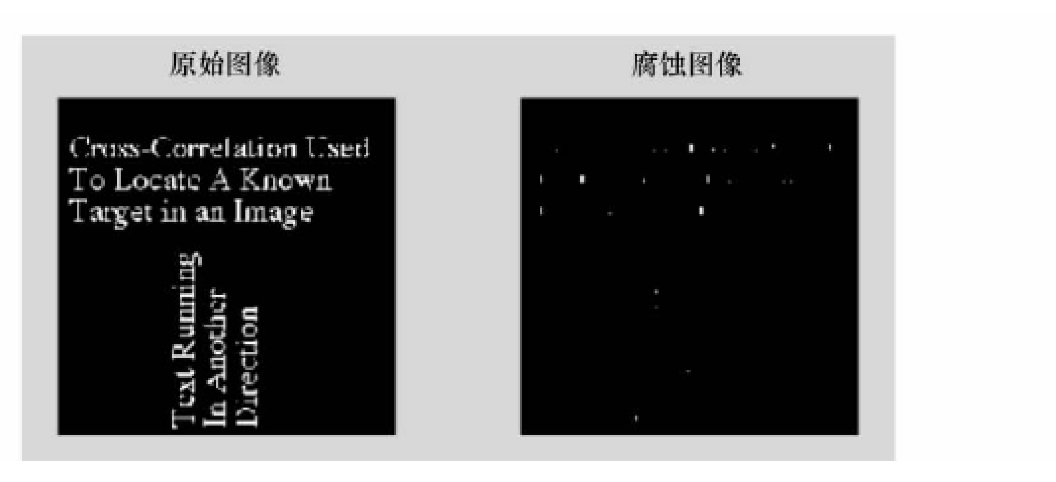


图 8.9 图像和腐蚀图像的对比

3. 开启和闭合

利用 Matlab 图像处理工具箱中的 bwmorph 函数来作开启和闭合运算：

函数：bwmorph。

功能：对图像作指定的形态运算。

语法格式：

```
BW2 = bwmorph(BW1,operation)
BW2 = bwmorph(BW1,operation,n)
```

说明：

BW2 = bwmorph(BW1,operation)对图像 BW1 作 operation 指定的形态学运算。

BW2 = bwmorph(BW1,operation,n) 对图像 BW1 作 n 次 operation 指定的形态学运算。n 可以取 Inf，在这种情况下，操作将持续到图像不再变化为止。在这些函数中，BW1 可以取 double 型或者 unit8 型。

operation 参数的类型和功能见表 8.1。

表 8.1 operation 参数

operation 参数	功能
bothat	闭包运算。先腐蚀，再膨胀，然后减去原图
erode	用结构元素 ones(3)作腐蚀运算
shrink	n=Inf，将对象收缩成点。消除像素，使没有孔的对象收缩成为一点，使有孔对象收缩成为外层边缘，在每个孔间缩成一个相连的环
bridge	连接运算，将原来不相连的像素连接起来

（续）

operation 参数	功　　能
fill	填充运算，填充孤立的内部像素点
skel	抽取骨架运算，n=Inf，消除对象边缘上的点，但不导致图像分裂，这样得到了图像的骨架
clean	去掉孤立的亮点
hbreak	断开 H 形连接的像素
spur	去掉小短枝像素
close	进行二值闭运算(先膨胀，再扩张)
majority	如果像素的 8 个邻域中有 5 个以上像素为 1，则像素值为 1，否则为 0
thicken	在图像的外部不断增加像素，但保证增加的像素不会导致原来不连接的对象成为 8-连接，n=Inf
diag	使用对角线填充来消除背景的 8-连接
open	开启运算(先腐蚀，然后膨胀)
thin	n=Inf，将对象细化成为线。如果是无孔的对象，则缩成最小连接的一笔；如果是有孔的对象，则缩成外层边缘，而在每个孔间缩成相连的环
dilate	用元素 ones(3)膨胀
remove	去掉内部像素点，即若像素的四邻域都为 1，则像素值为 0，就使得边缘的像素是亮点
tophat	用原图减去开启运算后的图

【例 8.4】对图像 text.tif 作开启和闭合运算。

```
I=imread('circbw.tif');
I1=bwmorph(I,'open');
I2=bwmorph(I,'close');
subplot(1,3,1);
imshow(I);title('原始图像');
subplot(1,3,2);
imshow(I1);title('开启运算图像');
subplot(1,3,3);
imshow(I2);title('闭合运算图像');
```

其显示结果如图 8.10 所示。

开启和闭合这两种运算都可以除去比结构元素小的特定图像细节，同时保证不产生全局几何失真。开启运算可以把比结构元素小的突刺滤掉，切断细长连接而起到分离作用。闭合运算可以把比结构元素小的缺口或孔填充上，连接短的间断而起到连通的作用。

开启和闭合运算不受原点位置的影响，无论原点是否包含在结构元素中，开启和闭合的结果都是一样的。根据膨胀和腐蚀的对偶性可知，开启与闭合运算也具有对偶性。



图 8.10 开启和闭合运算图像

开启和闭合可以从图像中提取与其结构元素相匹配的形状，用 B 开启 A 就是选出了 A 中的某些与 B 相匹配的点，这些点可以由完全包含在 A 中的结构元素 B 的平移得到。

4. 细化与骨架提取

细化方法就是通过细化用骨架来代表对像的形状。所谓细化，就是把输入的具有一定宽度的图像轮廓用逐次去掉边缘的方法最终变为宽度仅为一个像素的骨架。

通过细化可以显示图像的拓扑结构。细化通过两步腐蚀来实现：

(1) 有条件的正常腐蚀，那些被标为除去的像素点并不立刻被除去；

(2) 只去掉那些消除后不破坏连通性的点。

抽取骨架是与细化有关的一种运算，也称为中轴变换或焚烧草技术。中轴是所有与物体在两个或更多非邻接边界点处相切的圆心的轨迹。但抽取骨架很少通过在物体内拟和圆来实现。在概念上，中轴可设想成按如下方式形成：想象一片与物体形状相同的草，沿其外围各点同时点火，火势向内蔓延，向前推进的火线相遇处各点的轨迹就是中轴。

【例 8.5】用形态学算子去掉图像 circles.tif 的内点，抽取骨架和细化。

```
BW1 = imread('circles.tif');
imshow(BW1);
%用形态学算子去掉图像的内点
BW2 = bwmorph(BW1,'remove');
%用形态学算子抽取图像的骨架
BW3 = bwmorph(BW1,'skel',Inf);
%用形态学算子细化图像
BW4= bwmorph(BW1,'thin', Inf);
subplot(2,2,1);
imshow(BW1,);title('原图像');
subplot(2,2,2);
```

```
imshow(BW2);title('去除内点图像');
subplot(2,2,3);
imshow(BW3);title('图像骨架');
subplot(2,2,4);
imshow(BW4);title('图像细化');
```

其显示结果如图 8.11 所示。

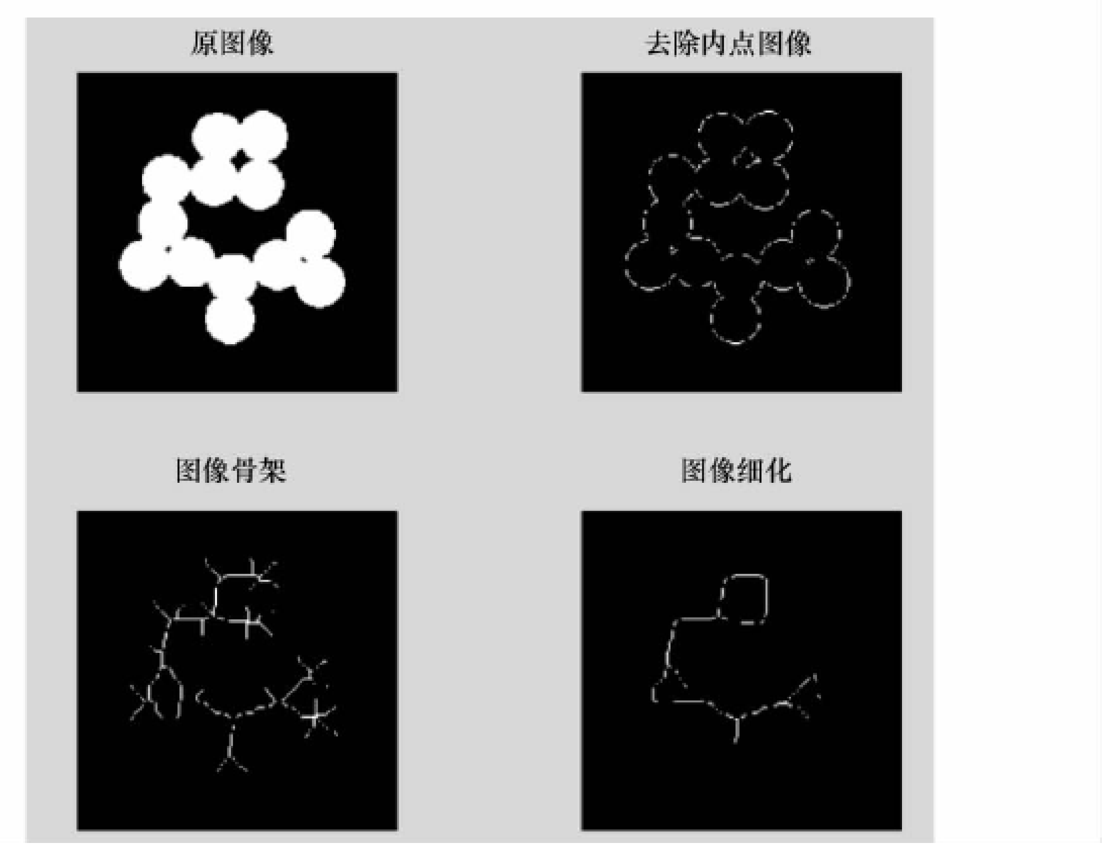


图 8.11　去除内点、细化和抽取骨架

5. 基于膨胀和腐蚀的形态操作

除了上面介绍的函数外，Matlab 图像处理工具箱还提供了其他一些用于完成基于膨胀和腐蚀操作的形态操作函数，见表 8.2。

表 8.2　基于膨胀和腐蚀操作的形态函数

函数	功能	说明
imopen	开启操作	BW=imopen(BW1,SE),SE 是结构元素
imclose	闭合操作	BW=imclose(BW1,SE),SE 是结构元素
imtophat	高帽变换	BW=imtophat(BW1,SE),从原始图像中减去开启计算后的图像，可以用来增强图像的对比度
imbothat	低帽变换	BW=imbothat(BW1,SE),从原始图像中减去关闭计算后的图像，可以用来寻找图像中的灰度谷值
bwhitmiss	逻辑“与”操作	BW=bwhitmiss(BW1,SE1,SE2),该函数的意义等于 & imerode(~BW1,SE2)

【例 8.6】对图像 rice.tif 做高帽变换。

说明：高帽变换可以增强图像的对比度，体现了原始图像中的灰度峰值。

```
I = imread('rice.tif');
subplot(1,3,1);
imshow(I), title('原始图像');
se = strel('disk',12);
J = imtophat(I,se); %利用结构元素,进行高帽变换
subplot(1,3,2);
imshow(J);title('高帽变换的结果');
%为了提高图像的可视化,进行直方图变换
K = imadjust(J,stretchlim(J));
subplot(1,3,3);
imshow(K);title('对比度增强的结果');
```

其显示结果如图 8.12 所示。

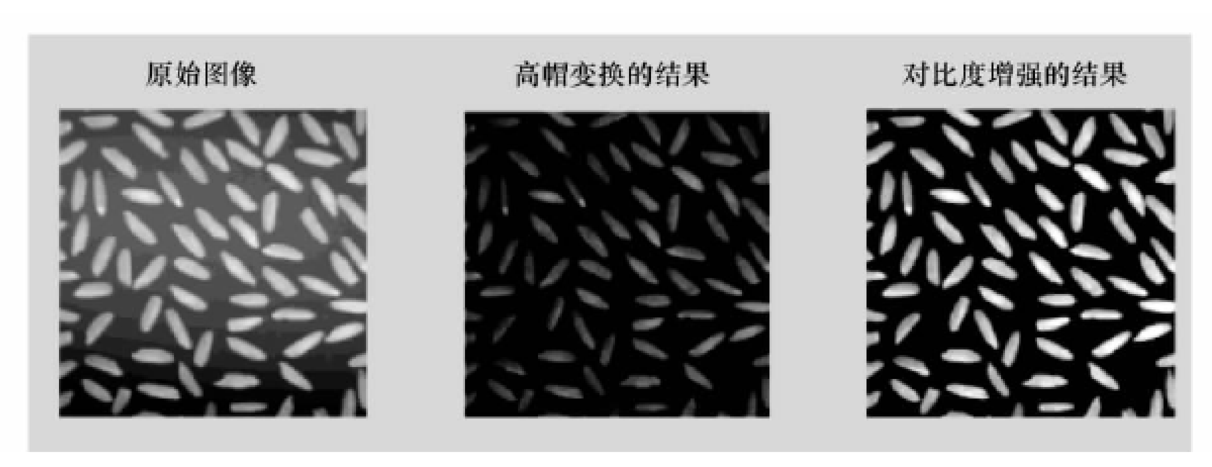


图 8.12　图像的高帽变换

高帽变换和低帽变换综合使用，可以用来提高图像的对比度。

8.2　二值图像及其特征提取

二值图像是一种所有像素值只能在两种可能的离散值中取一的图像，也称为黑白图像。本质上，这两个可取的像素值分别对应关闭(off)和打开(on)，也可记作 0,1。在 Matlab 中，二值图像用一个由 0 和 1 组成的二值矩阵表示，1 表示该像素处于前景，0 表示该像素处于背景。以这种方式来操作图像可以更容易识别出图像的结构特征。例如，在二值图像中可以非常容易地从图像背景中识别对象。

8.2.1　二值图像的生成

二值图像操作只返回与二值图像的形式或结构相关的信息，如果希望对其他类型的图像进行同样的操作，则首先要将其转换为二进制的图像格式。在 Matlab

图像处理工具箱中提供了函数 im2bw 来实现该转换。

函数：im2bw。

功能：把图像转换为二值图像。

语法格式：

```
BW = im2bw(I,level)
BW = im2bw(X,map,level)
BW = im2bw(RGB,level)
```

【例 8.7】把图像 trees 转换为二值图像。

```
load trees;
BW = im2bw(X,map,0.4);
subplot(1,2,1);
imshow(X,map);title('原图像');
subplot(1,2,2);
imshow(BW);title('二值图像');
```

其显示结果如图 8.13 所示。

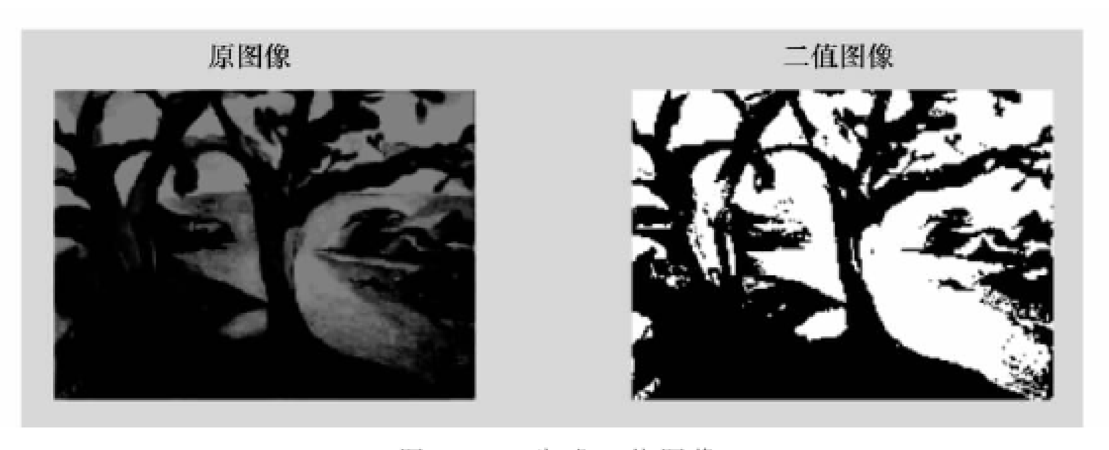


图 8.13　生成二值图像

可以看到转换为二值图像后，图像中只有黑白两种颜色，所以导致图像中许多细节都消失了。

8.2.2　特征提取

1. 图像面积

在进行图像处理时，会希望获得图像中改变某些特征的信息，例如，膨胀和腐蚀从定量的角度上来看就是二值图像中各对象面积的增大或者缩小。

Matlab 图像处理工具箱提供了 bwarea 用来计算二进制图像的面积，面积粗略地说就是图像中前景的像素的个数。但是 bwarea 函数并不是仅仅简单地计算非 0 像素的数目，而是在计算面积的过程中，对不同的像素赋予不同的权值，这个加权的过程是为了补偿用离散图像代表连续图像误差的过程。例如，在图像中一个 50 个像素的对角线比 50 个像素的水平线长。因此，经过加权，bwarea 函数返

回50个像素长的水平面积为50，而一个50个像素长的对角线为62.5。

函数：bwarea。

功能：获取二值图像的面积。

语法格式：

```
area=bwarea(BW)
```

【例8.8】计算图像circbw.tif在膨胀运算前后图像面积的改变。

```
BW = imread('circbw.tif');
SE = ones(5);
BW1 = imdilate(BW,SE);
subplot(1,2,1);imshow(BW);
subplot(1,2,2);imshow(BW1);
increase = (bwarea(BW1)- bwarea(BW))/bwarea(BW)
increase =
0.3456
```

其显示结果如图8.14所示。

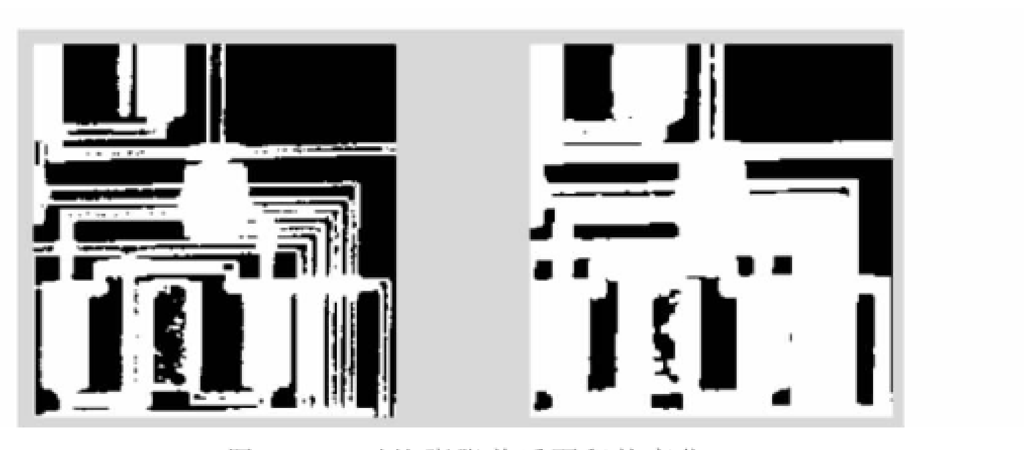


图8.14　对比膨胀前后面积的变化

计算图像中某个区域的面积以及这个区域的周长，根据它们的比值分析该区域所代表的图像形状，这是一种很常用的分析方法。

2. 欧拉数运算

在几何理论中，欧拉数是对图像拓扑的估计。欧拉数等于图像中所有对象的总数减去这些对象中孔洞的数目。

函数：bweuler。

功能：计算图像的欧拉数。

语法格式：

```
eul = bweuler(BW,N)
```

说明：

N表示连通类型，可以用4-连接或8-连接来进行计算，其默认值为8。

【例8.9】计算图像circles.tif的欧拉数。

```
BW = imread('circles.tif');
imshow(BW);
bweuler(BW)
结果如下：
ans =
    -2
```

其显示结果如图 8.15 所示。

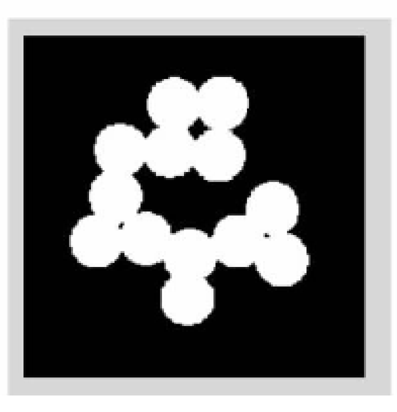


图 8.15　原始图像

在这个图像中，欧拉数的值是一个负数，表示图像中孔洞的数目大于对象的数目。利用欧拉数进行聚类分析是模式识别中一种常用的方法。

8.3　基于对象的操作

8.3.1　对象及边沿连接方式

在二值图像中，所谓的对象就是值为 1 的像素按照一定规则连接在一起的集合，值为 0 的像素代表的是背景。

图 8.16 所示的矩阵就代表了一个有 3×3 方形对象的二值图像。

```
0  0  0  0  0  0  0  0  0  0
0  1  1  1  0  0  0  0  0  0
0  1  1  1  0  0  0  0  0  0
0  1  1  1  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
```

图 8.16　对象示例

像素的连通性定义了该像素是与哪些像素相连接的，即像素的邻域是由哪些像素构成的。用户选择的邻域类型将影响图像中所能找到的对象数目和对象边界。许多形态操作的结果通常因指定的连接类型不同而不同。

连接是像素间的基本关系，在不同连接方式的预定下，同一个二值图像识别的对象不同。除了边缘点外，每个像素都有 8 个连接点。Matlab 中的二值图像连接

规则主要有两种:4-连接边沿和 8-连接边沿。4-连接边沿规则规定:只有垂直和水平方向的连接点可以成为连接像素;8-连接边沿规则规定:所有 8 个连接点均可以成为连接像素。如图 8.17 所示。

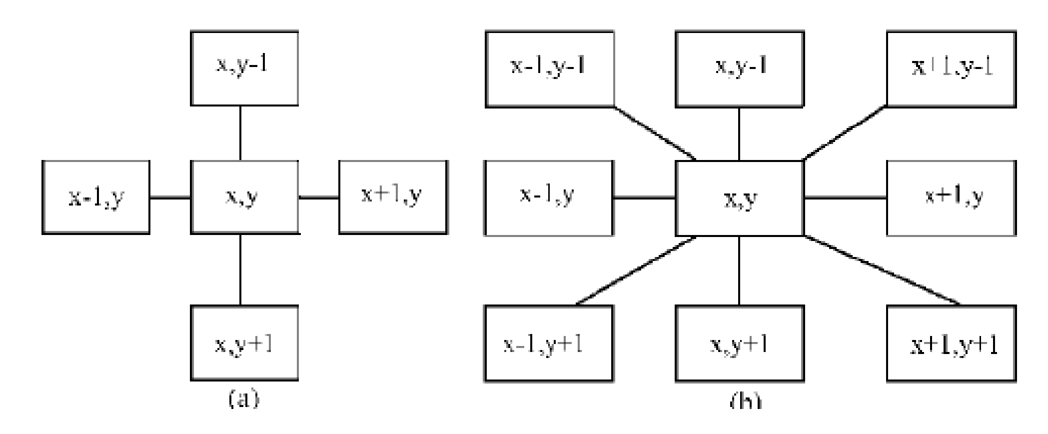


图 8.17 像素的连接关系

(a) 4-连接边沿;(b) 8-连接边沿。

在三维中也要定义像素的连接。在 Matlab 中定义了几种连接类型,见表 8.3。

表 8.3 连接类型

维数	连通性	说明	图例
二维	4-连接	边缘相连的像素为连接像素,即只有水平或垂直方向相连的相邻像素才能被视为同一对象	
	8-连接	边缘或角相连的像素为连接像素,即水平、垂直或对角方向相连的相邻像素都被看为同一对象	
三维	6-连接	面相连的像素为连接像素	
	18-连接	面或边相连的像素为连接像素	
	26-连接	面、边或角相连的像素为连接像素	

【例 8.10】求出以下图像矩阵在 8-连接和 4-连接规定下的对象数。

图像矩阵：

```
0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 1 1
0 0 0 0 0 0 0 1 1
```

在 4-连接边沿约定下图像包含 3 个对象：

```
0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 0 0 2 2 0 0 0 0
0 0 0 2 2 0 0 3 3
0 0 0 0 0 0 0 3 3
```

在 8-连接边沿约定下图像包含 2 个对象：

```
0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 2 2
0 0 0 0 0 0 0 2 2
```

8.3.2 对象标记和选择

1. 对象标记

在 Matlab 图像处理工具箱中，可以通过 bwlabel 函数来对二值图像中的对象进行标记。该函数返回的是与输入图像相同的矩阵，每一个对象对应一个整数。

函数：bwlabel。

功能：对二值图像中各个分离部分进行标记。

语法格式：

```
L = bwlabel(BW,n)
[L,num] = bwlabel(BW,n)
```

说明：

L 是返回的经过标记的图像；n 是区域的连接数，$n=4$ 表示采用 4-连接定义，$n=8$ 表示采用 8-连接定义，n 的默认值为 8；num 为 BW 中的连接对象数。

【例 8.11】标记下面图像矩阵中的对象。

```
BW = [1    1    1    0    0    0    0    0
      1    1    1    0    1    1    0    0
      1    1    1    0    1    1    0    0
      1    1    1    0    0    0    1    0
      1    1    1    0    0    0    1    0
      1    1    1    0    0    0    1    0
      1    1    1    0    0    1    1    0
      1    1    1    0    0    0    0    0]
L = bwlabel(BW,4)
```

得到以下标记结果：

```
L =
    1    1    1    0    0    0    0    0
    1    1    1    0    2    2    0    0
    1    1    1    0    2    2    0    0
    1    1    1    0    0    0    3    0
    1    1    1    0    0    0    3    0
    1    1    1    0    0    0    3    0
    1    1    1    0    0    3    3    0
    1    1    1    0    0    0    0    0
```

示例图像为：

```
imshow(L,'no truesize')
```

其显示结果如图 8.18 所示。

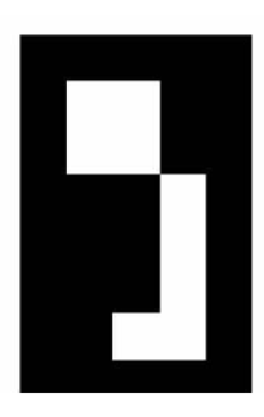


图 8.18　标记对象的显示

2. 对象选择

二值图像中，对象是指值为 1 且连接在一起的像素集合，函数 bwselect 用于在二值图像中选择特定的对象。Matlab 提供了 bwselect 函数来实现二值图像中的对象选择。

函数：bwselect。

功能：在二值图像中选择对象。

语法格式：

```
BW2 = bwselect(BW1,c,r,n)
BW2 = bwselect(BW1,n)
[BW2,idx] = bwselect(...)
```

说明：

bwselect 函数通过指定的像素，返回在输入图像中包含指定像素对象的二值图像。

BW2=bwselect(BW1,c,r,n)返回一个包含了像素(r,c)交叠的二值图像。r、c 可以是标量或者等长的向量。如果 $\boldsymbol{r}$、$\boldsymbol{c}$ 是向量，则BW2返回任何包含($r(k)$,$c(k)$)像素的物体。n 为区域的连接数，$n=4$ 表示4-连接，$n=8$ 表示8-连接，n 的默认值是8。

BW2=bwselect(BW1,n)用交互的方式确定提取对象的起始坐标。如果忽略BW1，则在当前的坐标系中选择。单击鼠标左键选定一个起始点，鼠标右键、空格表示删除一个点，Enter 键表示选择结束。

[BW2,idx] = bwselect(...)返回对象点数的线性下标。

【例 8.12】提取文本中的字符对象。

```
BW1 = imread('text.tif');
c = [16 90 144];
r = [85 197 247];
BW2 = bwselect(BW1,c,r,4);
subplot(1,2,1);imshow(BW1);
subplot(1,2,2);imshow(BW2);
```

其显示结果如图 8.19 所示。

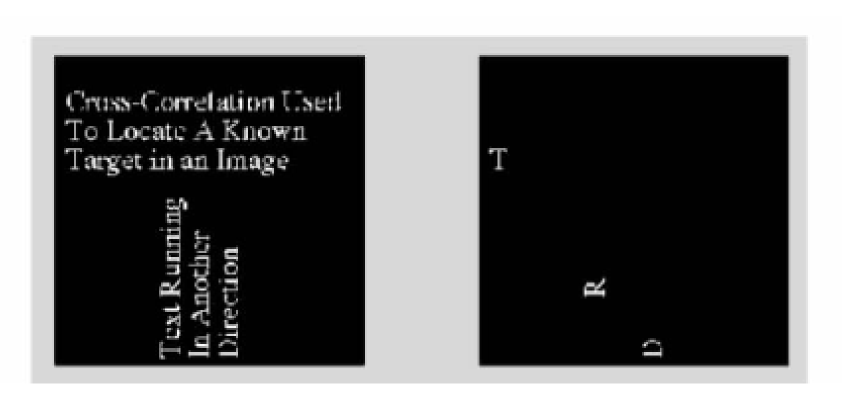


图 8.19　提取对象

3. 每个对象的特征

imfeature 函数也可以用于计算每个标注对象的特征，但与 bwarea 函数不同的是，imfeature 函数得到的面积是图像中像素的实际数目。

函数：imfeature。

功能:返回标注矩阵 **L** 中的每个标注对象的特征。

语法格式:

```
stats = imfeature(L,measurements)
stats = imfeature(L,measurements,n)
```

说明:

L 是标注矩阵;stats 是一个结构数组,显示了由 measurements 指定的每个对象的不同特征;*n* 表示选取的连接方式,只有在 measurements 取值"FilledImage"、"FilledArea"、"EulerNumber"时用到;measurements 是一个可以用逗号隔开的字符串列表,表示要计算哪些特征。

【例 8.13】计算每个标注对象的特征。

```
BW = imread('text.tif');
L = bwlabel(BW);
stats = imfeature(L,'all');
stats(23);
```

得到的结果如下:

```
ans = Area: 89
     Centroid: [95.6742 192.9775]
     BoundingBox: [87.5000 184.5000 16 15]
MajorAxisLength: 19.9127
MinorAxisLength: 14.2953
     Eccentricity: 0.6961
     Orientation: 9.0845
ConvexHull: [13x2 double]
ConvexImage: [15x16 logical ]
ConvexArea: 205
     Image: [15x16 logical ]
FilledImage: [15x16 logical ]
FilledArea: 122
EulerNumber: 0
     Extrema: [ 8x2 double]
EquivDiameter: 10.6451
     Solidity: 0.4341
     Extent: 0.3708
     PixelList: [89x2 double]
```

4. 种子填充

填充操作是一种通过像素边界来求取对象的操作,种子填充与其他操作的不同处在于它是对背景像素进行操作,而不是对前景像素进行操作。如果前景是 4-

连接的，则背景是8-连接的。

填充操作分三步进行：

（1）指定填充操作的连接性；

（2）指定一个背景点作为起始点；

（3）根据连接规则在二值图像中将与像素连接的背景点值由0变为1，在灰度图像中将被较亮区域围绕的黑暗区域的灰度值设置为与围绕区域的像数值相同的数值。

在Matlab图像处理中，以前是通过bwfill函数来提供操作，但是这个函数是过时的，在将来的版本中可能会被去掉，由imfill函数代替。

函数：imfill。

功能：对图像进行填充。

语法格式：

```
BW2 = imfill(BW)
[BW2,LOCATIONS]= imfill(BW)
BW2 = imfill(BW,LOCATIONS)
BW2 = imfill(BW,LOCATIONS,CONN)
BW2 = imfill(BW,'holes')
BW2 = imfill(BW,CONN,'holes')
```

说明：

BW1为输入的二进制图像；LOCATIONS表示填充的起始点；CONN表示使用的连接规则，默认值为4；参数holes表示填充输入二进制图像。

【例8.14】填充二值图像enamel.tif中的空洞。

```
I = imread('enamel.tif');
I2 = imcomplement(imfill(imcomplement(I),'holes'));
subplot(1,2,1);
imshow(I);title('原图像');
subplot(1,2,2);
imshow(I2);title('填充图像');
```

其显示结果如图8.20所示。

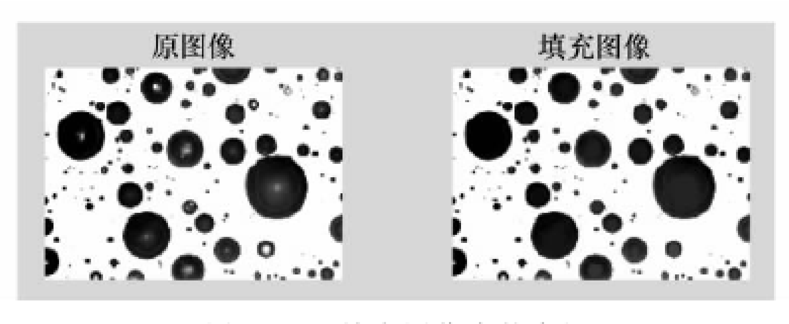


图8.20　填充图像中的空洞

从图 8.20 可以看出，imfill 函数删除了没有连接到边界的局部极小值，这个操作对于删除图像中的人工痕迹非常有用。填充操作也常常用来填充一幅图像中的“洞”。

8.3.3 边界标记

边界检测在一些实际应用中有着很重要的作用，Matlab 图像处理工具箱提供了 bwperim 函数来检测二进制图像的边界。边界的定义如下：

(1) 像素的值为 1；

(2) 像素邻域中至少有一个像素的值是 0。

像素的邻域可以采用 4-连接或者 8-连接规则。

函数：bwperim。

功能：对二值图像进行边界提取。

语法格式：

```
BW2 = bwperim(BW1,n)
```

说明：

BW1 表示输入的二进制图像；BW2 表示返回的 BW1 的边界图像。n 为邻域的类别，$n=4$ 表示采用 4-连接规则，$n=8$ 表示采用 8-连接规则，n 的默认值为 8。

【例 8.15】标记二值图像 blood1.tif 的边界。

```
BW = imread('blood1.tif');
BW1=im2bw(BW);
BW2 = bwperim(BW1,8);
subplot(1,2,1);
imshow(BW1);title('原始图像');
subplot(1,2,2);
imshow(BW2);title('边界提取图像');
```

其显示结果如图 8.21 所示。

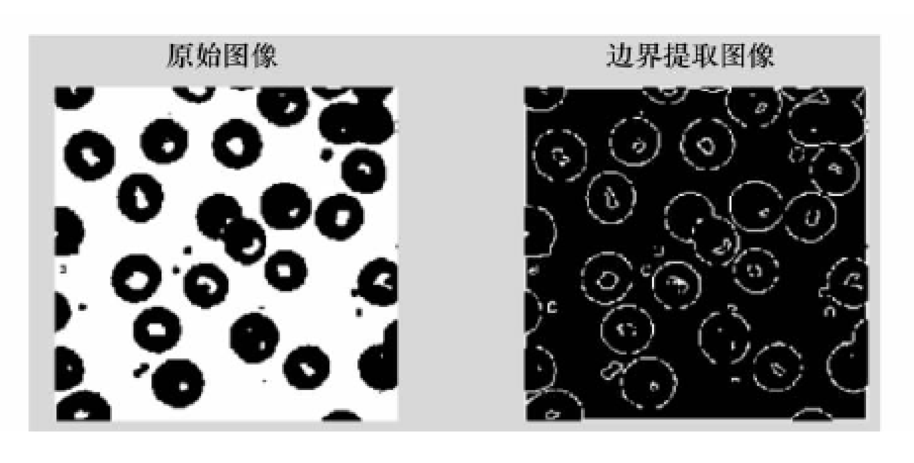


图 8.21　二值图像边界标记

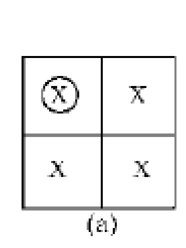
8.4 形态学应用

8.4.1 查找表操作

为了方便地进行二进制图像操作，Matlab 提供了一个非常有用的工具——查找表(Look-U PTable,LUT)，使用查找表可能会提高运算速度。

查找表就是将经过某一函数邻域操作后像素所有可能的计算结果都记录下来，在进行其他像素处理时直接通过查表得到像素的取值，而不是重复计算。一个查找表就是一个向量，其每个元素均表示边沿中的一种可能的像素组合的返回。

Matlab 提供的 makelut 函数可以为 2×2 和 3×3 的边沿创建查找表，applylut 函数则可以利用查找表对图像进行 2×2 和 3×3 的邻域(见图 8.22)运算。



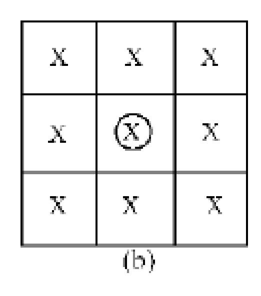


图 8.22 查找表邻域

(a) 2×2 邻域；(b) 3×3 邻域。

在 2×2 邻域中，元素可以有 16 种排列组合，所以 2×2 邻域的查找表是一个包含 16 个元素的列向量；在 3×3 邻域中，元素可以有 512 种排列组合，所以 3×3 邻域的查找表是一个包含 512 个元素的列向量。对于超过 3×3 邻域的查找表，如 4×4 邻域有 6536 种组合，在这种情况下利用查找表就没有什么实际意义了。

函数：makelut。

功能：为 2×2 和 3×3 邻域的边沿创建查找表。

语法格式：

```
lut = makelut(fun,n)
lut = makelut(fun,n,P1,P2,...)
```

说明：

fun 是一个返回标量的计算函数，此函数以 2×2 或 3×3 的 0、1 矩阵作为输入；n 取值为 2 或 3，表示 fun 的输入值的尺寸；$P1$，$P2$，…是函数 fun 的参数；返回值 lut 是一个 double 型向量。

makelut 函数通常和 applylut 函数联合使用。

函数：applylut。

功能：对二值图像执行邻域操作，生成对 lut 的索引矩阵，然后用 lut 中的实际

值替换索引的值。具体的算法取决于所采用的邻域。

语法格式：

```
A = applylut(BW,LUT)
```

说明：

BW 是输入的二进制图像；LUT 是由 makelut 函数返回的向量。

【例 8.16】通过查找表修改图中包含的文本。

```
%编写一个计算函数 f,如果一个 3×3 邻域内有三个或者三个以上的像素值为 1,则
%返回 1,否则返回 0。
f = inline('sum(x(:)) >= 3');
%用函数 f 创建一个查找表
lut = makelut(f,3);
BW1 = imread('text.tif');
BW2 = applylut(BW1,lut);
subplot(1,2,1);
imshow(BW1);title('原始图像');
subplot(1,2,2);imshow(BW2);title('查找表修改图像');
```

其显示结果如图 8.23 所示。



图 8.23　查找表修改图像

8.4.2　形态重构

形态重构是图像形态处理的一个重要操作，通常用来强调图像中与掩模图像指定对象相一致的部分，同时忽略图像中的其他对象。形态重构根据一幅图像（称为掩模图像）的特征对另外一幅图像（称为标记图像）进行重复膨胀，重点是要选择一个合适的标记图像，使膨胀所得的结果能够强调掩模图像中的主要图像。每一次膨胀处理从标记图像的峰值点开始，整个膨胀过程一直重复，直到图像的像素值不再变化为止。形态重构是基于膨胀操作的运算，具有以下三个独有的特征。

(1) 形态重构处理是基于两幅图像的：标记图像（Marker）和掩模图像（Mask）；而不仅仅是一幅图像和一个结构元素。

(2) 重构操作将重复进行直至图像稳定（即图像不再变化）。

(3) 形态重构是基于连接性概念的,而不是基于结构元素的。

Matlab 图像处理工具箱提供了 imreconstruct 函数来进行图像重构。

函数:imreconstruct。

功能:对图像进行重构。

语法格式:

```
IM = imreconstruct(MARKER,MASK)
IM = imreconstruct(MARKER,MASK,CONN)
```

说明:

MARKER 表示标记图像;MASK 表示掩模图像;CONN 是一个可选参数,用来指定连接类型,二维图像的默认值是 8-连接,三维图像的默认值是 26-连接。

为了更好地理解形态重构,看一下图 8.24 所示的简单图像矩阵,该图像包含了两个主要对象,这两个对象的像素块分别由数值 14 和 18 组成,图像的大部分背景像素值为 10,有一些背景像素为 11。

```
A = [10  10  10  10  10  10  10  10  10  10;
     10  14  14  14  10  10  11  10  11  10;
     10  14  14  14  10  10  10  11  10  10;
     10  14  14  14  10  10  11  10  11  10;
     10  10  10  10  10  10  10  10  10  10;
     10  11  10  10  10  18  18  18  10  10;
     10  10  10  11  10  18  18  18  10  10;
     10  10  11  10  10  18  18  18  10  10;
     10  11  10  11  10  10  10  10  10  10;
     10  10  10  10  10  10  11  10  10  10];
```

图 8.24 图像矩阵

对这个图像进行重构运算,将进行以下两步操作。

第一步 创建一个标记图像。就像膨胀和腐蚀中的结构元素一样,标记图像的特征决定了形态重构特征。标记图像的峰值应该指出想在掩模图像中强调的对象。一种创建标记图像的方法是用 imsubtract 函数从掩模图像中减去一个常数。例如:

```
marker = imsubtract(A,2)
marker =      8    8    8    8    8    8    8    8    8    8
              8   12   12   12    8    8    9    8    9    8
              8   12   12   12    8    8    8    9    8    8
              8   12   12   12    8    8    9    8    9    8
              8    8    8    8    8    8    8    8    8    8
              8    9    8    8    8   16   16   16    8    8
              8    8    8    9    8   16   16   16    8    8
              8    8    9    8    8   16   16   16    8    8
              8    9    8    9    8    8    8    8    8    8
              8    8    8    8    8    8    9    8    8    8
```

第二步用 imreconstruct 函数对图像进行重构操作：

```
recon = imreconstruct(marker, mask)
```

其显示结果如图 8.25 所示。

```
recon =
    10    10    10    10    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

图 8.25　重构图像结果

从图 8.25 中可以看到，除了峰值外，其他的灰度波动都被去掉了。这样图像仅仅包含原始图像的两个主要对象，并且背景色的灰度也一致。

图 8.26 说明了形态重构的过程。图像重构运算可以看做是在掩模的限制下，对标记图像进行连续膨胀。当膨胀操作不会再改变图像数值时，重构过程停止，最后一次的膨胀结果就是重构图像。

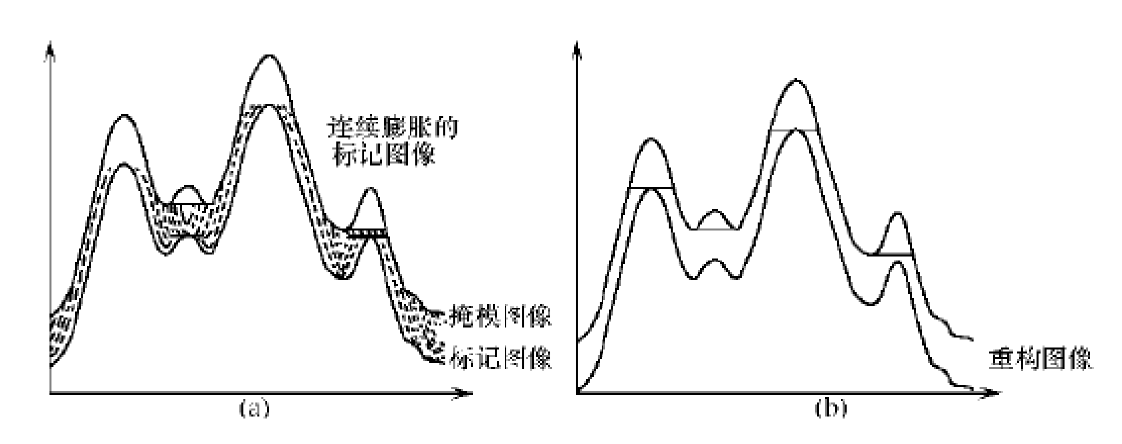


图 8.26　形态重构过程示意图

(a) 图像重构过程；(b) 图像重构结果。

8.4.3　距离变换

距离变换可以提供一个图像像素点的距离估计矩阵，根据该矩阵可以进行图像分割。

Matlab 图像处理工具箱提供了函数 bwdist 用来计算二进制图像中每一个像素值为 0 的像素与其最近非零像素间的距离。

函数：bwdist。

功能：返回图像的距离矩阵。

语法格式：

```
D = bwdist(BW)
```

```
[D,L]= bwdist(BW)
[D,L]= bwdist(BW,METHOD)
```

说明：

BW 为输入图像；***D*** 为输出的距离矩阵，表示这点的像素与其最近非零像素间的距离；***L*** 是输出的索引矩阵，表示距离这个像素点最近的非零像素的索引；METHOD 表示计算中所采用的距离矩阵的类型，取值可以是“euclidean”、“cityblock”、“chessboard”、“quasi-euclidean”（见表 8.4），默认的类型是“euclidean”。

表 8.4　各种距离的定义

距离矩阵	描述	在二维情况下的公式	图像和距离变换矩阵
欧氏矩阵 euclidean	欧氏距离是指两个像素间的直线距离	$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$	原图像： 0 0 0 0 1 0 0 0 0 距离变换： 1.41 1.0 1.41 1.0 0.0 1.0 1.41 1.0 1.41
城市矩阵 cityblock	城市块距离矩阵根据 4-连接邻域估计像素间的路径。边缘相接的像素间隔 1 个单位，对角相连的像素间隔 2 个单位	$\lvert x_1-x_2\rvert+\lvert y_1-y_2\rvert$	原图像： 0 0 0 0 1 0 0 0 0 距离变换： 2 1 2 1 0 1 2 1 2
棋盘矩阵 chessboard	棋盘距离矩阵根据 8-连接邻域估计像素间的路径。边缘或对角相接的像素间隔 1 个单位	$\max(\lvert x_1-x_2\rvert,\lvert y_1-y_2\rvert)$	原图像： 0 0 0 0 1 0 0 0 0 距离变换： 1 1 1 1 1 1 1 1 1
准欧氏矩阵 quasi-euclidean	准欧氏矩阵按照水平、垂直和对象集合分段估计全部的欧氏距离	$\lvert x_1-x_2\rvert+(\sqrt{2}-1)\lvert y_1-y_2\rvert$	原图像： 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 距离变换： 2.8 2.2 2.0 2.2 2.8 2.2 1.4 1.0 1.4 2.2 2.0 1.0 0 1.0 2.0 2.2 1.4 1.0 1.4 2.2 2.8 2.2 2.0 2.2 2.8

表 8.4 说明了距离矩阵的类型和算法。

【例 8.17】求图像的距离矩阵。

```
bw = zeros(5,5); bw(2,2) = 1; bw(4,4) = 1
[D,L]= bwdist(bw);
```

得到如下的结果：

```
D =
    1.4142    1.0000    1.4142    2.2361    3.1623
    1.0000         0    1.0000    2.0000    2.2361
    1.4142    1.0000    1.4142    1.0000    1.4142
    2.2361    2.0000    1.0000         0    1.0000
    3.1623    2.2361    1.4142    1.0000    1.4142
```

```
L =
    7     7     7     7     7
    7     7     7     7    19
    7     7     7    19    19
    7     7    19    19    19
```

【例 8.18】对图像的各种类型的距离矩阵进行比较。

```
bw = zeros(200,200); bw(50,50) = 1; bw(50,150) = 1;
bw(150,100) = 1;
D1 = bwdist(bw,'euclidean');
D2 = bwdist(bw,'cityblock');
D3 = bwdist(bw,'chessboard');
D4 = bwdist(bw,'quasi-euclidean');
figure
subplot(2,2,1), subimage(mat2gray(D1)), title('euclidean 距离')
hold on, imcontour(D1)
subplot(2,2,2), subimage(mat2gray(D2)), title('city block 距离')
hold on, imcontour(D2)
subplot(2,2,3), subimage(mat2gray(D3)), title('chessboard 距离')
hold on, imcontour(D3)
subplot(2,2,4), subimage(mat2gray(D4)), title('Quasi-Euclidean 距离')
hold on, imcontour(D4)
```

其显示结果如图 8.27 所示。

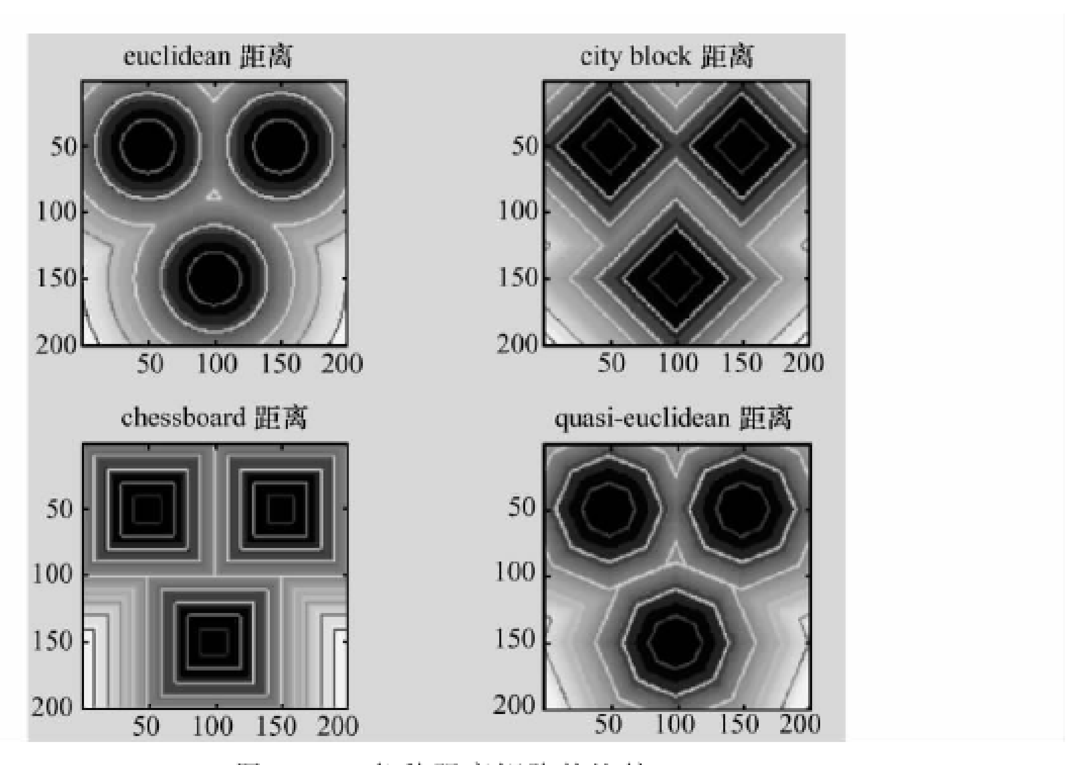


图 8.27 各种距离矩阵的比较

从图 8.27 中，可以看到：准欧氏矩阵和欧氏矩阵得到的结果图像最接近。

8.4.4 图像的极值处理方法

灰度图像可以用三维图像来看待，其中 x 轴和 y 轴表示像素的位置，而 z 轴表示每一个像素的亮度。在三维的解释下，像素的灰度值可以代表地图中的高度值，图像的高灰度值和低灰度值处就相当于地图的峰和谷。通常图像的峰和谷代表相关的图像像素，具有重要的形态特征。

例如，在有几个球形物体的对象中，高灰度的像素点可以表示对象的顶部。在形态操作中这些图像的极值可以被用来识别图像中的对象。

一幅图像可以拥有多个局部极小值和极大值，但是只能有一个全局最大值和最小值。找到图像的峰和谷可以用来创建在形态重构中使用到的标记图像。

图 8.28 说明了一维图像极大值、极小值、最大值和最小值的概念。在图中，x 轴代表像素位置，y 轴代表灰度值。

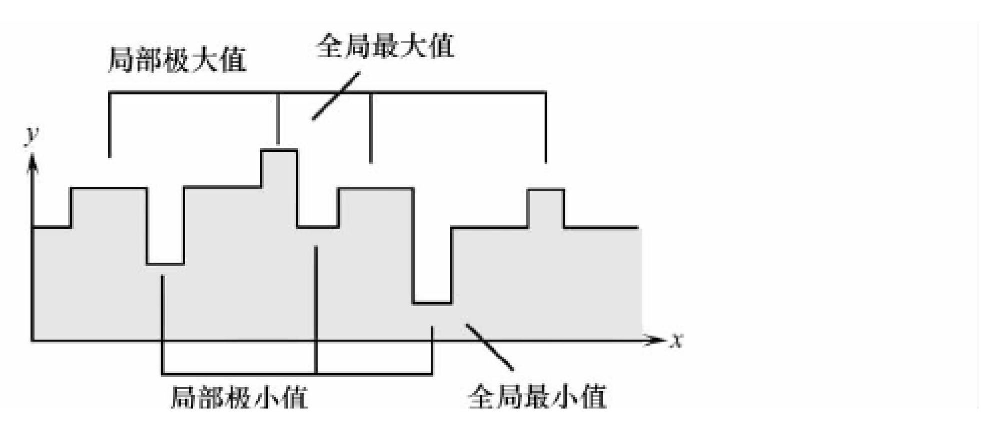


图 8.28 极值和最值图

Matlab 图像处理工具箱提供了一些函数来寻找图像中的极小值、极大值、最小值和最大值：

(1) imregionalmax 和 imregionalmin 函数用来寻找图像中的所有局部极大值和极小值；

(2) imextendedmax 和 imextendemin 函数用来寻找所有大于或小于指定值的局部极大值和极小值。

这些函数的输入是一个灰度图像，输出是一个二值图像，在二值图像中，极大值或极小值被设置为 1，其他像素值被设置为 0。

函数：imregionalmax。

功能：寻找局部极大值。

语法格式：

```
BW = imregionalmax(I)
BW = imregionalmax(I,CONN)
```

说明：

I 是输入的灰度图像；BW 是返回的二进制图像，局部极大值或局部极小值被设置为 1，其他像素则被设置为 0；CONN 表示连接类型。

【例 8.19】对图 8.29 所示简单图像的矩阵求解局部极大值和极小值。

```
A = [10   10   10   10   10   10   10   10   10   10;
     10   13   13   13   10   10   11   10   11   10;
     10   13   13   13   10   10   10   11   10   10;
     10   13   13   13   10   10   11   10   11   10;
     10   10   10   10   10   10   10   10   10   10;
     10   11   10   10   10   18   18   18   10   10;
     10   10   10   11   10   18   18   18   10   10;
     10   10   11   10   10   18   18   18   10   10;
     10   11   10   11   10   10   10   10   10   10;
     10   10   10   10   10   10   11   10   10   10]
```

图 8.29　原始图像矩阵

可以看到图像中包含 13 和 18 局部极大值的两个像素块，其他一些局部极大值为 11。

B = imregionalmax(A)

其显示结果如图 8.30 所示。

```
B=   0   0   0   0   0   0   0   0   0   0
     0   1   1   1   0   0   1   0   1   0
     0   1   1   1   0   0   0   1   0   0
     0   1   1   1   0   0   1   0   1   0
     0   0   0   0   0   0   0   0   0   0
     0   1   0   0   0   1   1   1   0   0
     0   0   0   1   0   1   1   1   0   0
     0   0   1   0   0   1   1   1   0   0
     0   1   0   1   0   0   0   0   0   0
     0   0   0   0   0   0   1   0   0   0
```

图 8.30　图的局部极大值矩阵

可以看到极大值像素被设置为 1，而其他像素被设置为 0。

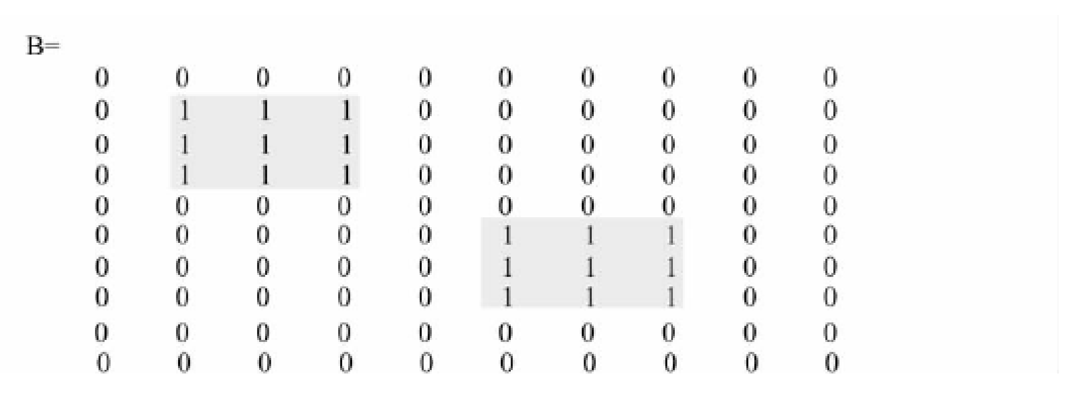
在有些情况下，可能只需要寻找那些灰度变化超过指定阈值的区域，即像素与其邻域像素间的灰度变化大于一个固定阈值的区域。这种情况下，可以通过 imextendedmax 函数来进行。

【例 8.20】找到图 8.29 所示矩阵中数值大于其邻域像素值两个单位的局部极大值。

B=imextendedmax(A,2)

其显示结果如图 8.31 所示。

在一幅图像中，每一小的灰度变化都会形成一个局部极大值和极小值。但是有时可能只对那些有重要意义的极大值和极小值感兴趣，而不需要那些由于背景



```
B=
   0   0   0   0   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   1   1   1   0   0
   0   0   0   0   0   1   1   1   0   0
   0   0   0   0   0   1   1   1   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
```

图 8.31　指定阈值的图像局部极大值矩阵

纹理形成的灰度变化较小的极大值或极小值。

为了去除这些不需要的极大值和极小值的影响，同时又要寻找到重要的极大值和极小值，可以使用 imhmax 函数或 imhmin 函数。

函数：

I2＝imhmax(I,H,CONN)

I2＝imhmin(I,H,CONN)

说明：

H 是指定的一个固定的标准或阈值，用于去处那些灰度变化小于 H 的极大值或大于 H 的极小值。

imhmax 函数和 imhmin 函数分别仅仅对极大值和极小值产生影响，对其他像素不起作用。

【例 8.21】寻找图像(见图 8.32)中灰度变化大于 2 的极大值。

从图像矩阵中可以看到，图像中有两个重要的局部极大值(像素值为 14 和 18 的块)和一些其他极大值(像素值为 11)。为了去掉那些不重要的极大值(像素值为 11)，可以使用函数 imhmax，指定阈值为 2。

```
A = [10   10   10   10   10   10   10   10   10   10;
     10   14   14   14   10   10   11   10   11   10;
     10   14   14   14   10   10   10   11   10   10;
     10   14   14   14   10   10   11   10   11   10;
     10   10   10   10   10   10   10   10   10   10;
     10   11   10   10   10   18   18   18   10   10;
     10   10   10   11   10   18   18   18   10   10;
     10   10   11   10   10   18   18   18   10   10;
     10   11   10   11   10   10   10   10   10   10;
     10   10   10   10   10   10   11   10   10   10];
```

图 8.32　原始图像

B＝imhmax(A,2)

其显示结果如图 8.33 所示。

从图 8.33 可以看出，两个重要的极大值尽管数值减小，但还是被保留了下来，而不重要的极值被去掉(11 变为 10)。

```
B=
    10    10    10    10    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    12    12    12    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    16    16    16    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

图 8.33　处理后的极大值

图 8.34 所示为图像矩阵中第二行的操作过程。

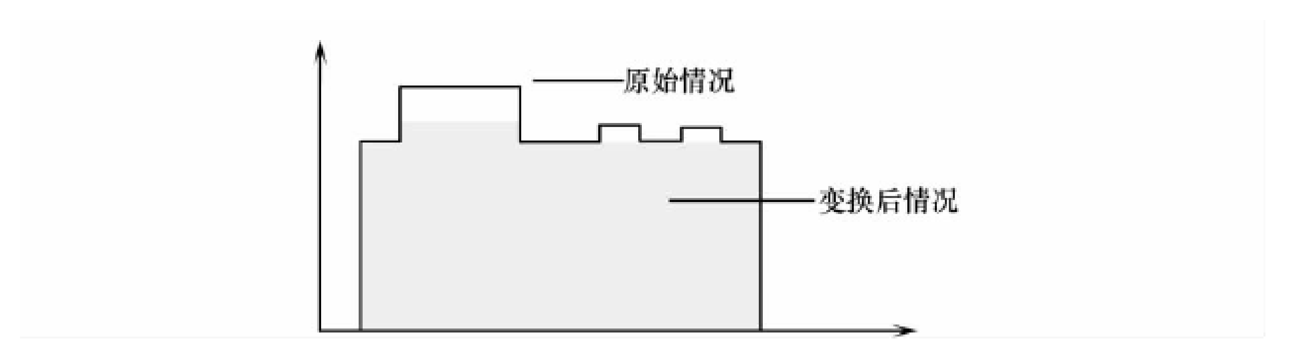


图 8.34　操作的过程

还可以使用 imimposemin 函数来强调图像中被指定的极小值。imimposemin 函数使用形态重构来消除图像中除指定极小值以外的其他所有的极小值，其语法格式如下：

```
I2 = imimposemin(I,BW)
```

说明：

I 为输入图像；BW 是一个与 I 大小相同的二进制图像，其不为 0 的元素指定极小值；CONN 表示连接的类型。

注意：

imregionalmin 函数、imregionalmax 函数、imextendedmin 函数和 imextendedmax 函数返回一个标记了图像极值位置的二值图像矩阵；imhmax 函数和 imhmin 函数返回一个修改后的图像矩阵。

【例 8.22】创建一个图像包含两个重要的局部极小值和其他一些局部极小值，并强调最重要局部极小值，删除其他极小值。

```
mask = uint8(10 * ones(10,10));
mask(6:8,6:8) = 2;
mask(2:4,2:4) = 7;
mask(3,3) = 5;
mask(2,9) = 9
```

```
mask(3,8) = 9
mask(9,2) = 9
mask(8,3) = 9
```

其显示结果如图 8.35 所示。

```
mask= 10  10  10  10  10  10  10  10  10  10
      10   7   7   7  10  10  10  10   9  10
      10   7   6   7  10  10  10   9  10  10
      10   7   7   7  10  10  10  10  10  10
      10  10  10  10  10  10  10  10  10  10
      10  10  10  10  10   2   2   2  10  10
      10  10  10  10  10   2   2   2  10  10
      10  10   9  10  10   2   2   2  10  10
      10   9  10  10  10  10  10  10  10  10
      10  10  10  10  10  10  10  10  10  10
```

图 8.35　创建的原始图像

首先创建一幅能够查明两个重要的极小值的标记图像。可以通过明确地设置像素来指定极小值，或者使用形态函数来抽取掩模图像中希望强调的特征。以下代码使用 imextendedmin 函数来获得一幅指明两个极小值位置的二进制图像。

```
marker = imextendedmin(mask,1);%创建标记图像
I = imimposemin(mask,marker)
I =

    11    11    11    11    11    11    11    11    11    11
    11     8     8     8    11    11    11    11    11    11
    11     8     0     8    11    11    11    11    11    11
    11     8     8     8    11    11    11    11    11    11
    11    11    11    11    11    11    11    11    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11     0     0     0    11    11
    11    11    11    11    11    11    11    11    11    11
    11    11    11    11    11    11    11    11    11    11
```

图 8.36 说明了 imimposemin 函数是如何修改图像第二行的外观的。在掩模图像中使用 imimposemin 函数在标记图像指定的点处创建新的极小值。imimposemin函数最终将标记图像指定的像素值设置为图像数据类型支持的极限值，并将修改图像中其他所有的像素值来删除其他的极小值。

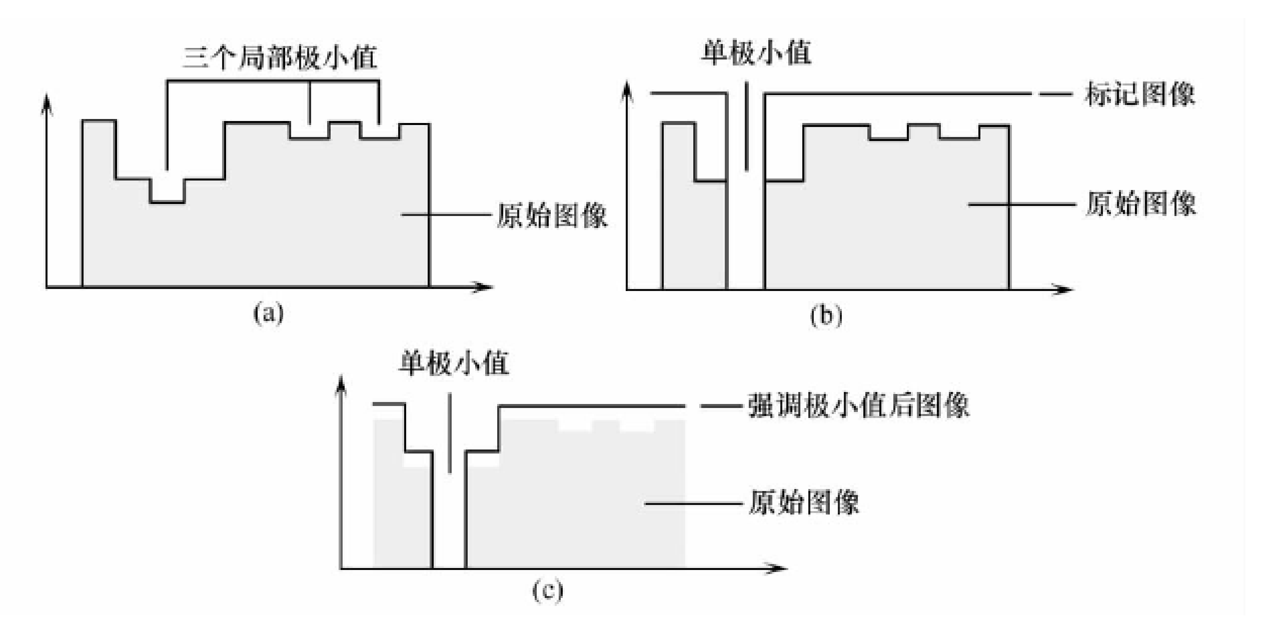


图 8.36　修改过程

第 9 章　综合实例

在实际应用中，如对钢纹的区域标识的实际应用中，有些问题不能只是通过仅仅一种图像处理的方法就能够取得满意的效果，而是必须综合应用多种方法，只有对这些方法进行综合应用才能够取得令人满意的效果。

9.1　光照不均的校正

如果图像是由光的反射形成的或者光源照射在景物上的光照不均匀，光照较强的部分将比较明亮，光照较弱的部分就比较暗。这种情况是很常见的，在这种情况下，对图像进行边缘检测、模式识别等工作时就不能取得令人满意的效果，所以首先必须对光照不均匀进行校正。

进行光照校正的处理一般按以下步骤进行：

(1) 首先估计图像背景的灰度，一般是取图像中每个 32×32 大小的图形块中的最小值作为图像背景的灰度；

(2) 然后将粗略估计出的背景灰度矩阵扩展成和原始图像大小相同的矩阵；

(3) 从原始图像中减去前两步计算出的背景灰度矩阵，校正光照的不均匀，但是此操作同时也会导致前景图像变暗；

(4) 通过调整图像的灰度校正。

【例 9.1】对光照不均图像 rice.tif(见图 9.1)的光照进行校正。

图 9.2 是图像背景灰度的一个统计图。



图 9.1　光照不均匀的原始图像

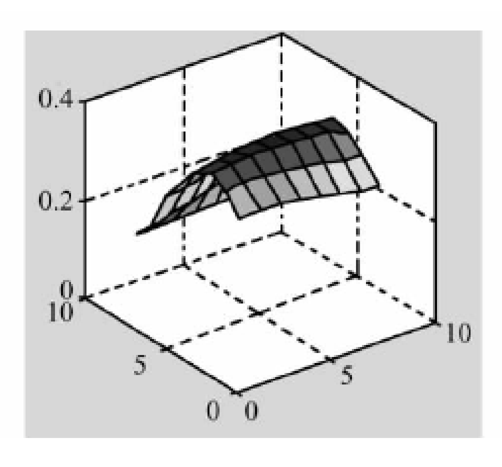


图 9.2　背景灰度的极小值

对图像进行观察，可以看到图像上部的背景比图像下部的背景要亮，即图像下部的灰度要比上部的灰度低，可以通过以下操作使整幅图像的背景色一致。

```
BW=imread('rice.tif');
subplot(2,2,1);imshow(BW);title('原始图像');
BW2=im2double(BW);
bg32=blkproc(BW2,[32 32],'min(x(:))');          %得到每个子块的极小值
bg256=imresize(bg32,[256 256],'bicubic');       %生成背景矩阵
subplot(2,2,2);imshow(bg256);title('背景灰度扩展结果');
d=BW2-bg256;
subplot(2,2,3);imshow(d);title('原始图像减去背景结果');
adjustbw=imadjust(d,[0 max(d(:))],[0 1],1);
subplot(2,2,4);imshow(adjustbw);title('最终处理结果');
```

其显示结果如图 9.3 所示。

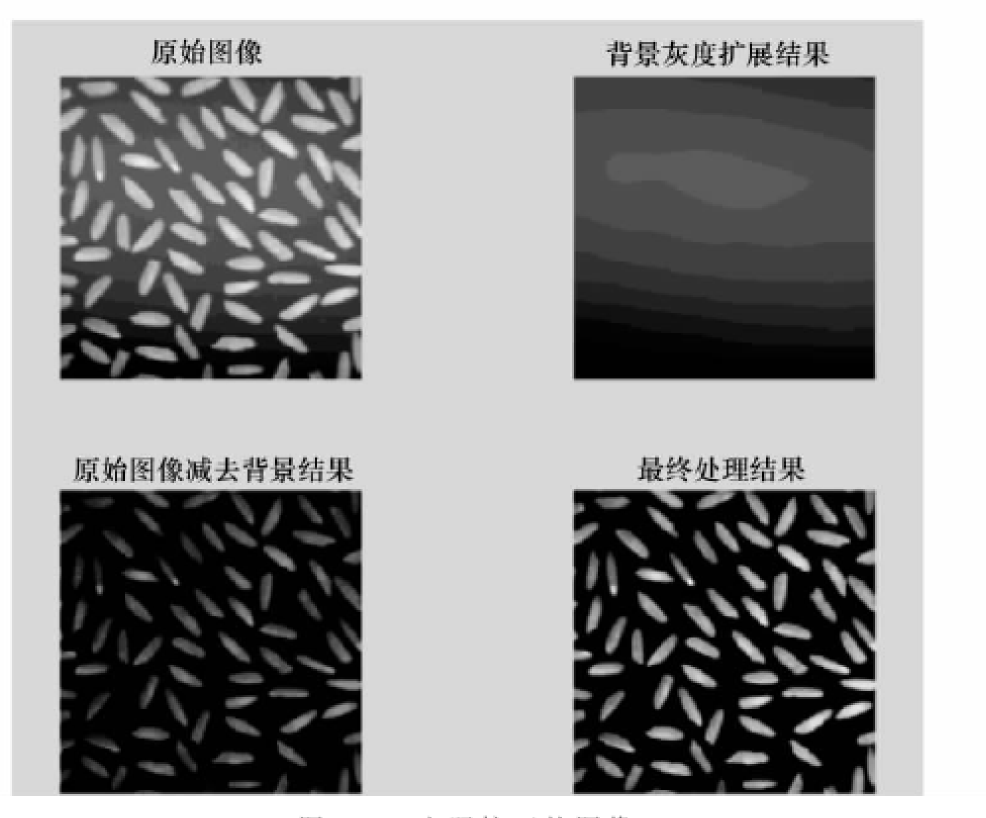


图 9.3　光照校正的图像

从图 9.3 的最终处理结果可以看出，同原始图像相比，图像背景的灰度分布均匀，米粒和背景间的对比增强，这样有利于进行图像分割、识别等操作。

9.2　基于特征的逻辑运算

在数字图像处理中运用基于特征的逻辑运算，可以实现图像的合成，从图像中提取物体等功能。

1. 基于特征的与运算

【例 9.2】找出图像 dots 和图像 box 相重合的对象。

寻找两幅图像的重合对象一般按以下步骤进行：

(1) 对两幅图像进行“与”操作，找出两幅图像中的公共部分；

(2) 将第一步的结果作为 bwselect 函数的输出，实现基于特征的“与”运算，其结果就是要求的对象。

代码如下：

```
load imdemos dots box
subplot(1,2,1);
imshow(box);title('box 图像');
subplot(1,2,2);
imshow(dots);title('dots 图像');
```

此程序显示的原始图像如图 9.4 所示。

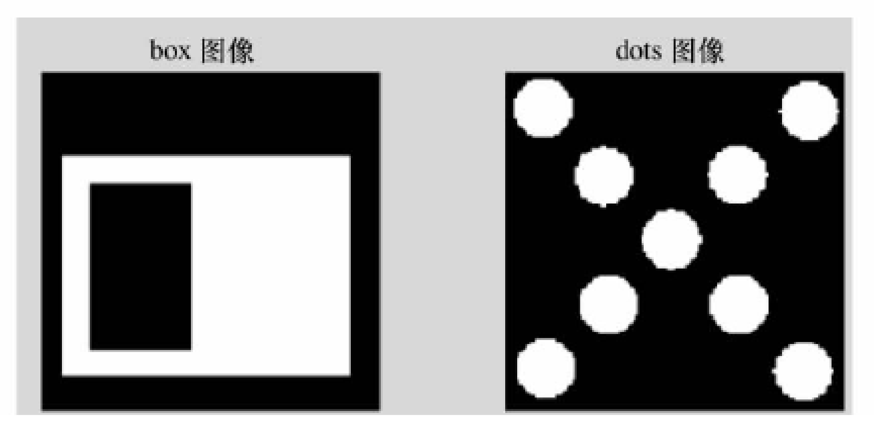


图 9.4　原始图像

```
logical_and=box & dots;                    %用与运算找出两幅图像的共同部分
subplot(1,2,1);imshow(logical_and);title('与图像');
[r,c]=find(logical_and);
%上面的代码完成了第一步
%将与运算的结果作为 bwselect 函数的输入,找出所要求的对象
feature_and=bwselect(dots, c, r);          %基于特征的与运算
subplot(1,2,2);
imshow(feature_and);title('基于特征的与的结果');
```

其显示结果如图 9.5 所示。

2. 利用逻辑运算提取物体

【例 9.3】辨识出哪些细菌包含一个或多个亮颗粒，确定包含颗粒的细菌的个数。

说明：这里有一些细菌可能包含多个颗粒。

算法如下：

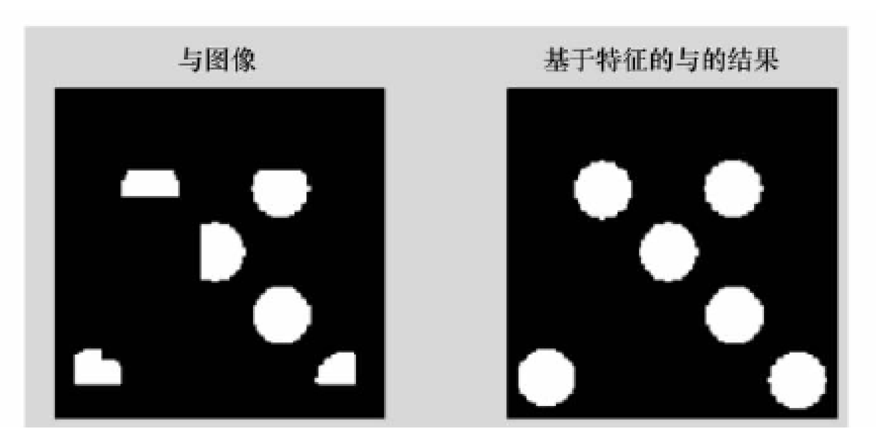


图 9.5　重合的对象

(1) 设置一个适当的阈值对原始图像进行分割,得到二值分割图像;
(2) 设置一个适当的阈值对原始图像进行高通滤波,得到图像中的亮点,但是这个图像可能受到噪声的干扰;
(3) 将(1)、(2)步的结果进行逻辑与运算,这样得到图像中的亮点(避免了噪声的干扰);
(4) 对第(1)步得到的二值分割图像进行腐蚀操作,这样将细菌独立出来并同时去除了细菌边缘的点;
(5) 利用基于特征的"与"运算提取出指定的物体。

```
load imdemos bacteria;
subplot(2,2,1);imshow(bacteria);title('细菌图像');
bact_bw = ~(bacteria>100);                    %设置阈值并对图像进行二值分割
subplot(2,2,2)
imshow(bact_bw);title('二值分割图像');
%上面的代码完成了第一步
filtered = filter2(fspecial('laplacian'), bacteria);  %对图像进行滤波处理
%上面的代码完成了第二步
bact_granules = (filtered >-4) & bact_bw;       %进行"与"处理
subplot(2,2,3);imshow(filtered, []);title('拉普拉斯滤波结果');
%上面的代码完成了第三步
subplot(2,2,4);imshow(bact_granules);title('有亮点的细菌');
%上面的代码完成了第四步
```

其显示结果如图 9.6 所示。

```
granules=imerode(bact_bw,ones(3,3)) & (bact_granules==0);
figure;imshow(granules);title('独立出来的亮点');
[r,c]=find(granules);
subplot(1,2,1);imshow(bacteria);title('细菌图像');
result=bwselect(bact_bw, c, r);
figure;subplot(1,2,2);imshow(result);title('含有亮点的细菌');
```

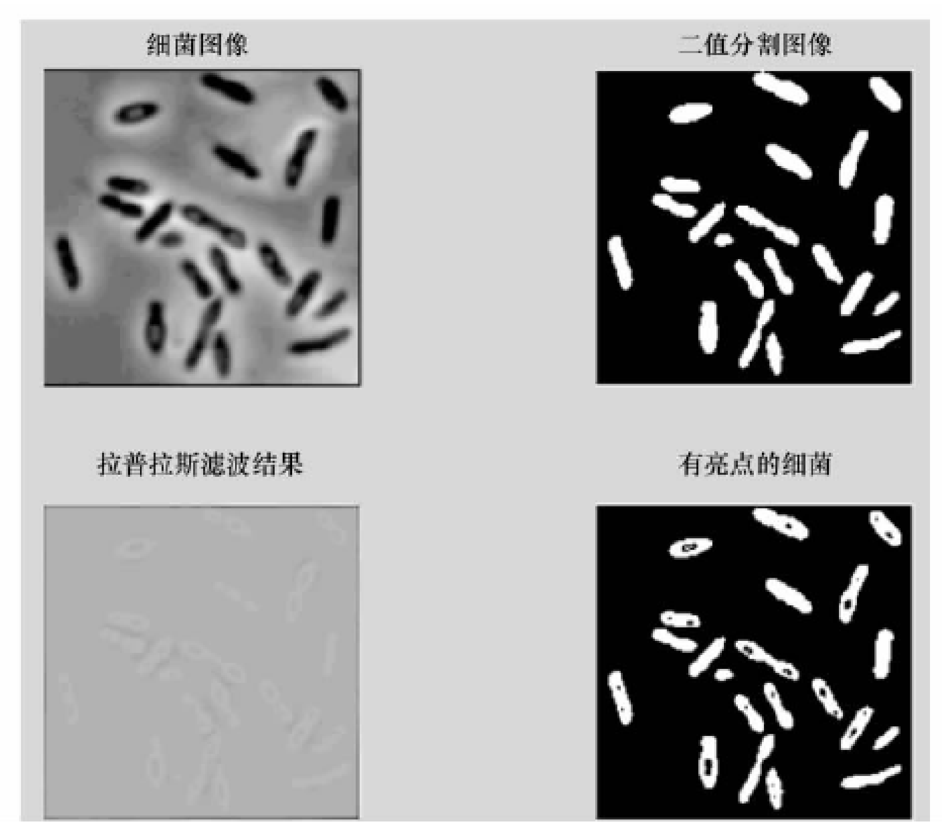


图 9.6　包含亮点的细菌

其显示结果如图 9.7 所示。

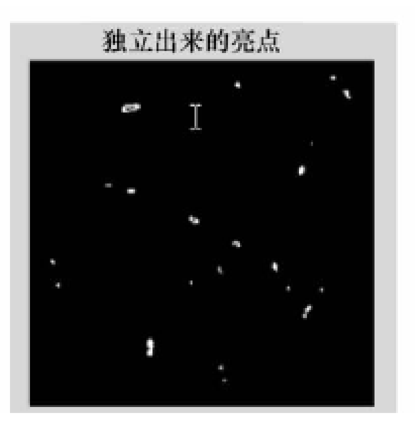


图 9.7　细菌中的亮点

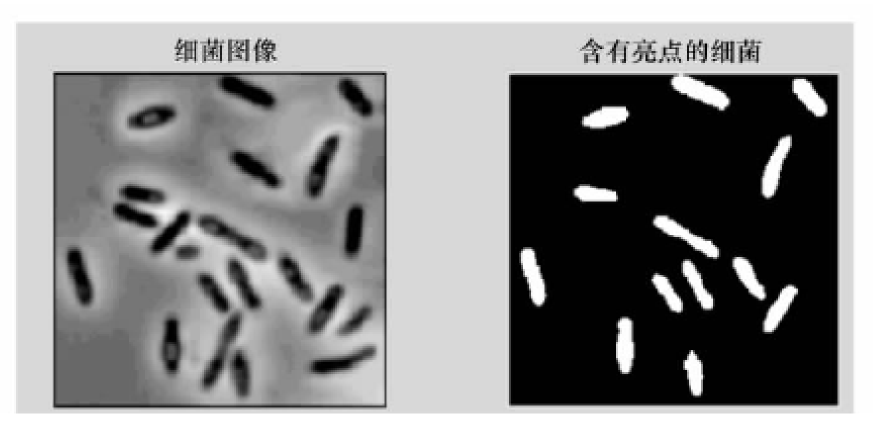


图 9.8　含有亮点的细菌

9.3 图像分割

1．对钢纹的区域标识

通过取两次阈值、基于特征的逻辑、二形态学和相连接成分的标识，确定钢的显微图像中的颗粒边界，标识不同的颗粒。

【例 9.4】标识钢的显微图像中的不同颗粒。

说明：由于图像中的杂质较多，颗粒的边缘有很多尖刺，所以要先对图像进行一些预处理，然后对处理后的图像进行标识。

算法如下：

(1) 除去图像中的较小区域；

(2) 除去图像中的尖刺像素；

(3) 标识图像。

钢的图像如图 9.9 所示。

```
load imdemos steel;imshow(steel);
```

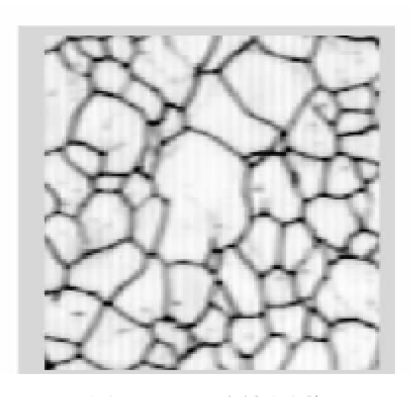


图 9.9　原始图像

第一步 对钢的图像取两个不同的阈值 70 和 210，处理代码如下：

```
bw_70=steel>70;              %取阈值 70 进行分割
subplot(1,2,1);
imshow(bw_70);
bw_210=steel>210;            %取阈值 210 进行分割
subplot(1,2,2);
imshow(bw_210);
```

其显示结果如图 9.10 所示。

第二步 用取阈值 70 的图像来选取取阈值 210 求反图像中的白色图像，除去图像中的较小区域。

```
[r,c] =find(bw_70==0);
bw_clean=bwselect(~bw_210, c, r, 8);
imshow(bw_clean);
```

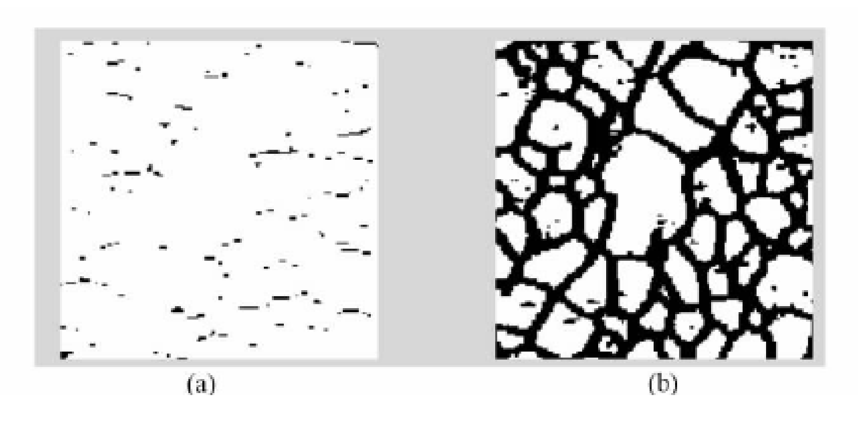


图 9.10 阈值分割的结果

(a) 用阈值 70 得到的结果；(b) 用阈值 210 得到的结果。

其显示结果如图 9.11 所示。

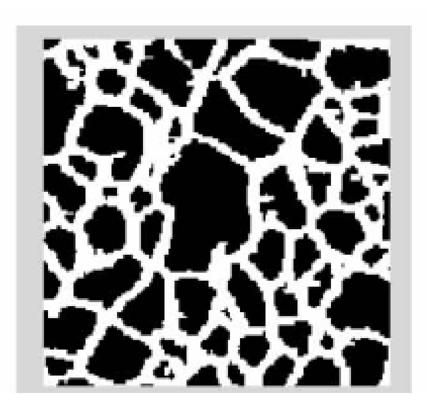


图 9.11 除去较小区域的求反图像

第三步 抽取去除了小区域图像的骨架，然后从图像的骨架中去除尖刺像素。

```
bw_skel=bwmorph(bw_clean, 'skel', 6);          %抽取图像骨架
subplot(1,2,1);
imshow(bw_skel);
bw_pruned=bwmorph(bw_skel, 'spur', 8);         %去除图像尖刺
subplot(1,2,2);imshow(bw_pruned);
```

其显示结果如图 9.12 所示。

第四步 对第三步的结果求逻辑非，就得到了颗粒的边界图。

```
grain_boundaries=~bw_pruned;                   %进行逻辑与运算
imshow(grain_boundaries);
```

其显示结果如图 9.13 所示。

第五步 将图像中的颗粒标识出来，并用伪彩色图像显示。

```
[labeled,N]=bwlabel(grain_boundaries,4);
colored = label2rgb(labeled);imshow(colored);
```

其显示结果如图 9.14 所示。

2. 检测图像中的微小结构

检测图像中的微小结构一般采取以下步骤：

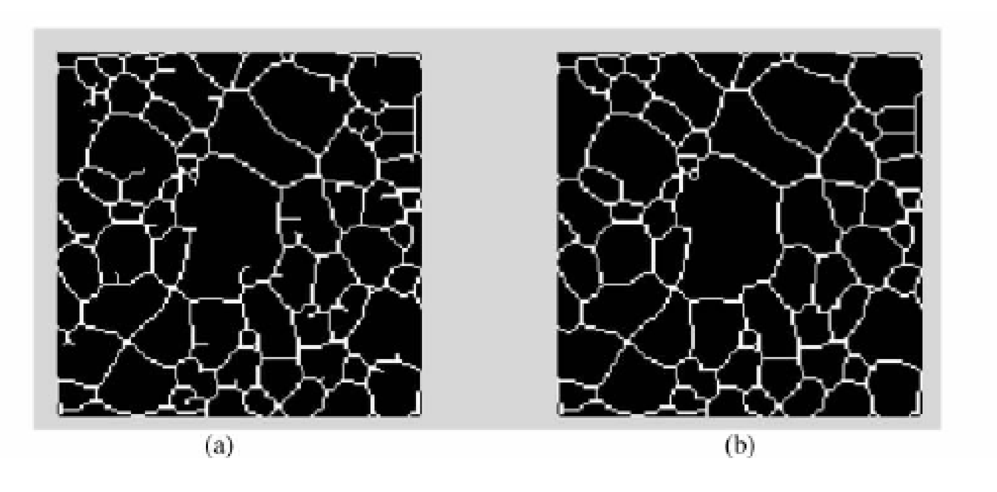


图 9.12　去除尖刺的图像骨架

(a) 图像骨架；(b) 去除尖刺的骨架。

(1) 对原始图像进行阈值分割，得到图像中微小的物体的图像；

(2) 对原始图像进行闭合、开启运算，得到图像中较大的物体的图像；

(3) 对(1)、(2)步的结果进行“与”操作，得到结果。

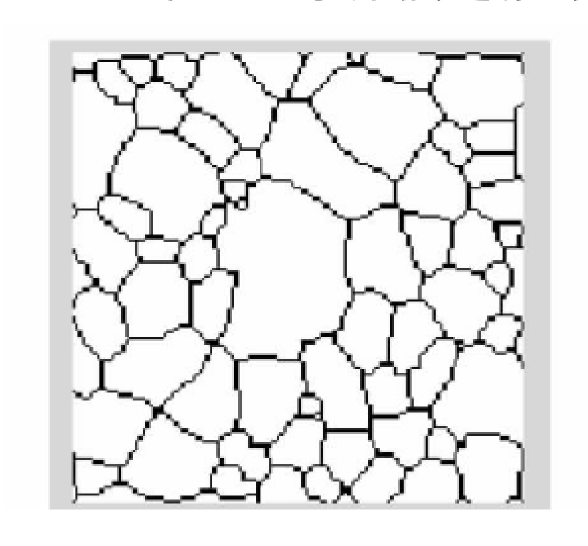


图 9.13　颗粒图像

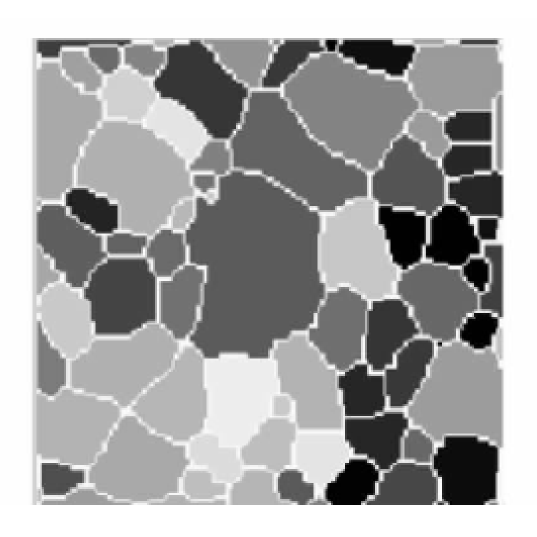


图 9.14　伪彩色显示

【例 9.5】检测出图像中的微小结构。

第一步得到图像变量。

```
I=imread('pearlite.tif');
figure, imshow(I), title('原始图像');
```

其显示结果如图 9.15 所示。

第二步调用 imcomplement 函数对图像求补，然后调用 im2bw 函数对图像进行阈值截取分割，获得图像中的较小物体。

```
Ic=imcomplement(I);                    %进行图像反色
subplot(1,2,1);
imshow(Ic);title('反色的图像');
BW = im2bw(Ic, graythresh(Ic));        %阈值分割,得到小对象
```

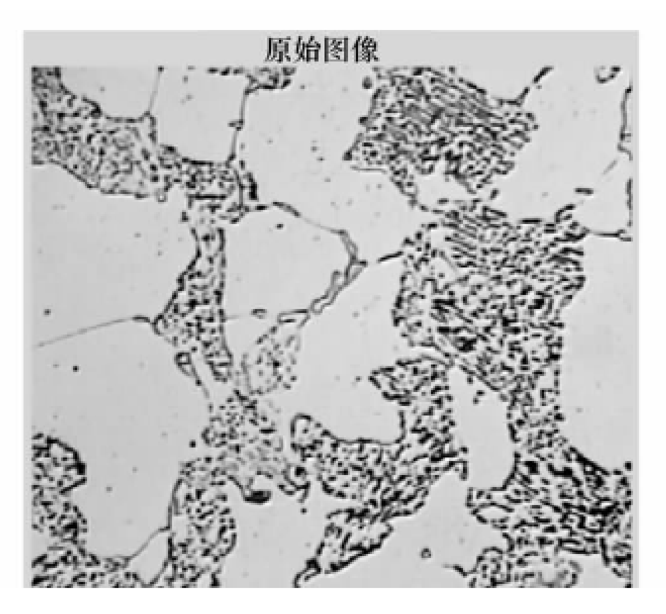


图 9.15 原始图像

```
subplot(1,2,2);
imshow(BW);title('阈值分割后的结果');
```

其显示结果如图 9.16 所示。

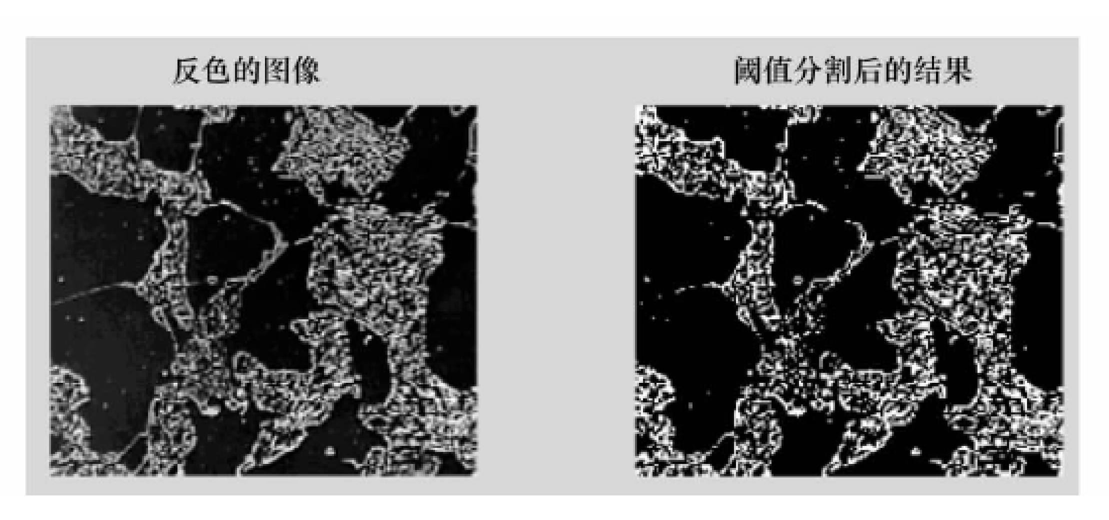


图 9.16 较小对象图像

第三步首先对图像进行闭合运算，然后进行开启运算，这样可以删除图像中某些较小的对象，从而得到较大物体的图像。

```
se=strel('disk', 6);                    %创建结构元素
BWc=imclose(BW, se);                    %进行闭合运算
BWco=imopen(BWc, se);                   %进行开启运算
subplot(1,2,1);
imshow(BWc);title('闭合运算后的图像');
subplot(1,2,2);
imshow(BWco);title('开启运算后的最终结果');
```

其显示结果如图 9.17 所示。

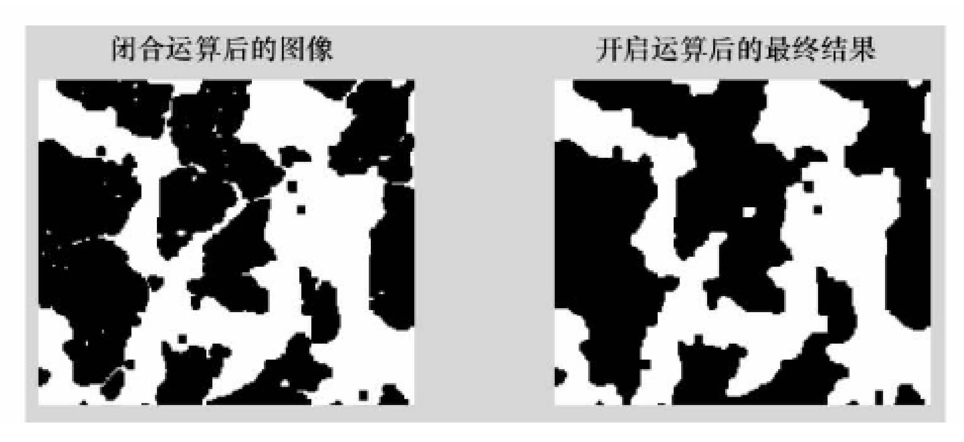


图 9.17　开启闭合运算结果

第四步将第二步得到的较小对象的图像和第三步得到的大对象的图像进行逻辑“与“的操作，这样就得到了原始图像中的所有小对象。

```
mask=BW & BWco;                    %进行逻辑“与”操作
figure, imshow(mask), title('检测结果');
```

其显示结果如图 9.18 所示。

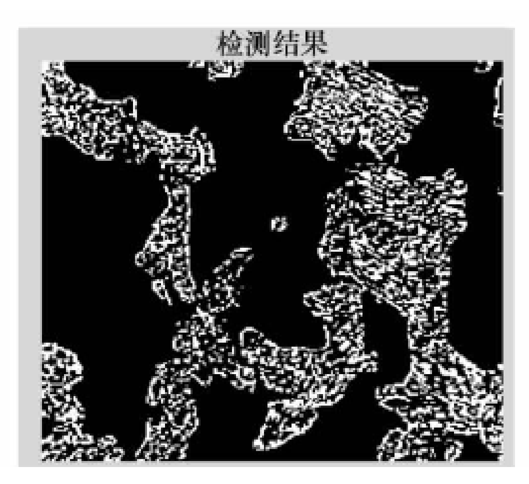


图 9.18　结果图像

3. 检测图像中相互接触的对象

相互接触的同一类对象，由于灰度接近，如果直接进行检测，不能取得很好的效果；所以要首先进行一些预处理，这样可以取得较好的效果。

检测的步骤如下：

(1) 利用高帽变换和低帽变换将图像中的对比度调大；

(2) 将灰度直方图中谷底的像素设置为 0；

(3) 用分水岭函数对图像进行分割。

【例 9.6】检测图像中相互接触的对象。

第一步读入图像变量，如图 9.19 所示。

```
afm=imread('afmsurf.tif');
figure, imshow(afm);
```

第二步增大对象中的对比度。从图 9.19 中可以看到，由于相互接触的各个对

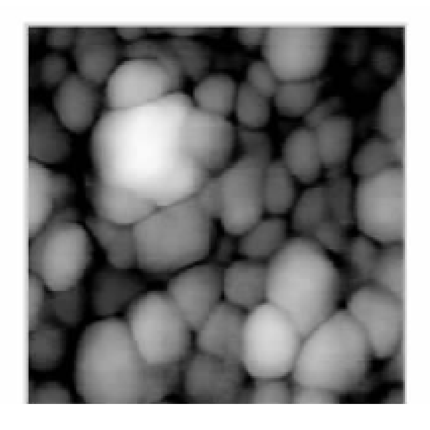


图 9.19 原始图像

象的灰度比较接近，所以进行图像分割前应该先增强对比度。调节相互接触的物体的对比度的最好的方法就是综合使用高帽变换和低帽变换。在程序中将用半径为 15 个像素的圆盘形结构元素进行图像的形态操作。

```
se=strel('disk', 15);                    %生成圆盘形结构元素
Ito p=imtophat(afm, se);                  %对原图像进行高帽变换
Ibot=imbothat(afm, se);                   %对原图像进行低帽变换
subplot(1,2,1); imshow(Itop, []); title('高帽变换图像');
subplot(1,2,2);imshow(Ibot, []); title('低帽变换图像');
```

其显示结果如图 9.20 所示。

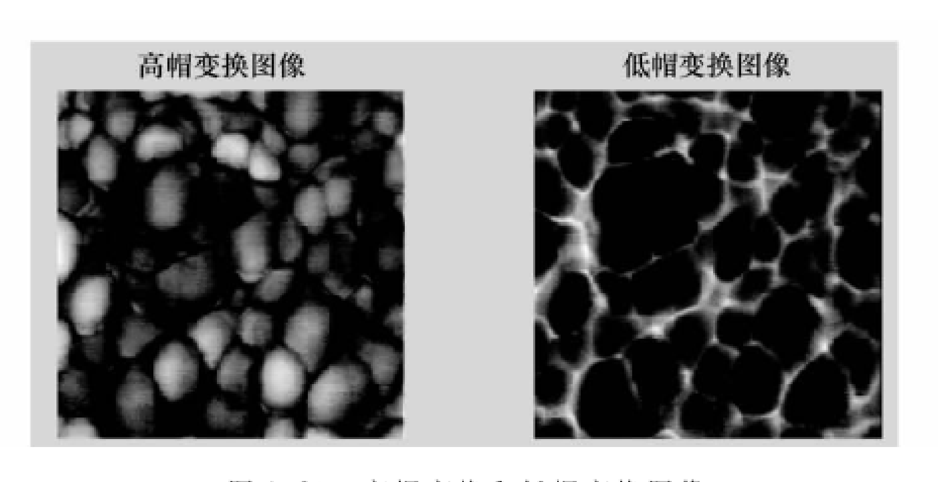


图 9.20 高帽变换和低帽变换图像

高帽变换是原始图像的开启计算的结果与原始图像的差，得到的图像体现了原始图像中的灰度峰值；低帽变换是原始图像的关闭计算的结果与原始图像的差，得到的图像体现了原始图像中的灰度谷值。为了增强最大对比度的图像，必须进行最准确的阈值分割，所以将高帽变换结果与原始图像相加后再与低帽变换结果相减。最后，为了便于观察将图像反色。

```
Ienhance=imsubtract(imadd(Itop, afm), Ibot);     %高帽图像和低帽图像的复合运算
figure, imshow(Ienhance), title('增强的图像');
```

其显示结果如图 9.21 所示。

```
Iec=imcomplement(Ienhance);          %对图像进行反色
figure, imshow(Iec), title('对增强图像的反色');
```

其显示结果如图 9.22 所示。

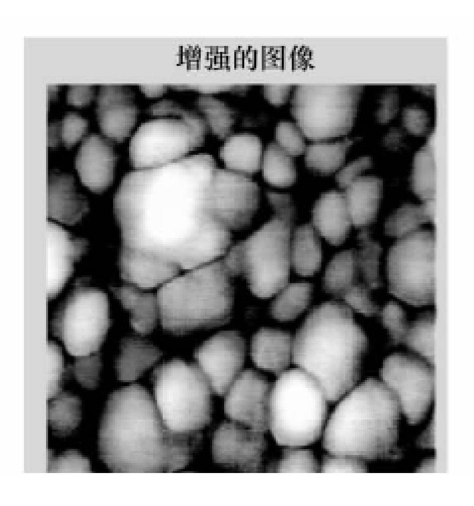


图 9.21　增强的图像

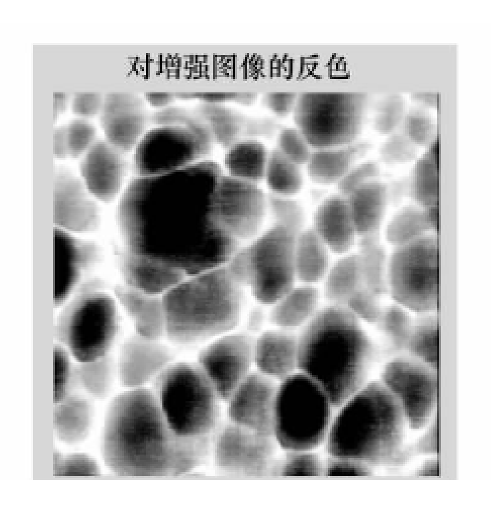


图 9.22　反色图像

第三步将直方图中,处于谷底的像素设置为 0。然后设置一个阈值,使用 imextendedmin 函数来搜索符合要求的图像的谷值,再调用 imimposemin 函数将图像中所有谷值像素设置为 0。

```
Iemin=imextendedmin(Iec, 22);          %得到极小值图像矩阵
Iimpose=imimposemin(Iec, Iemin);
subplot(1,2,1);
imshow(Iemin);title('图像谷值图');
subplot(1,2,2);
imshow(Iimpose), title('谷值像素设置为 0 图');
```

其显示结果如图 9.23 所示。

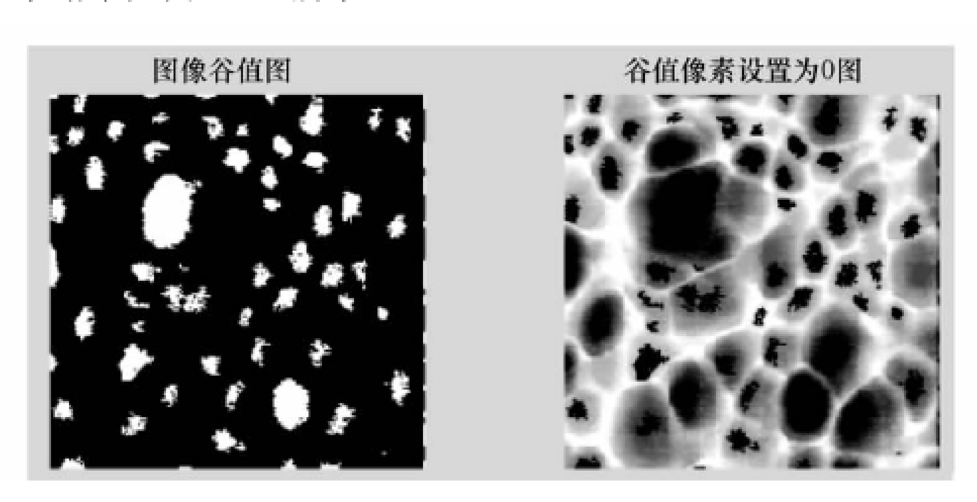


图 9.23　修改谷值的图像

第四步进行图像的分割。这里调用分水岭函数对图像进行分割。为了观察方便,将分割的结果用伪彩色图像显示。

```
wat=watershed(Iimpose);
```

```
rgb=label2rgb(wat);
figure, imshow(rgb);
title('分水岭函数分割图像');
```

其显示结果如图 9.24 所示。

第五步抽取分割图像中感兴趣的对象特征。

```
area=[stats(:).Area];
orient=[stats(:).Orientation];
figure, plot(area, orient, 'b*');
```

其显示结果如图 9.25 所示。

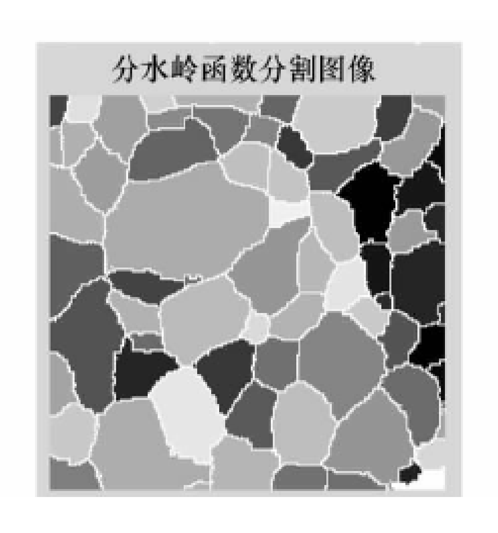



图 9.24　分割结果图像

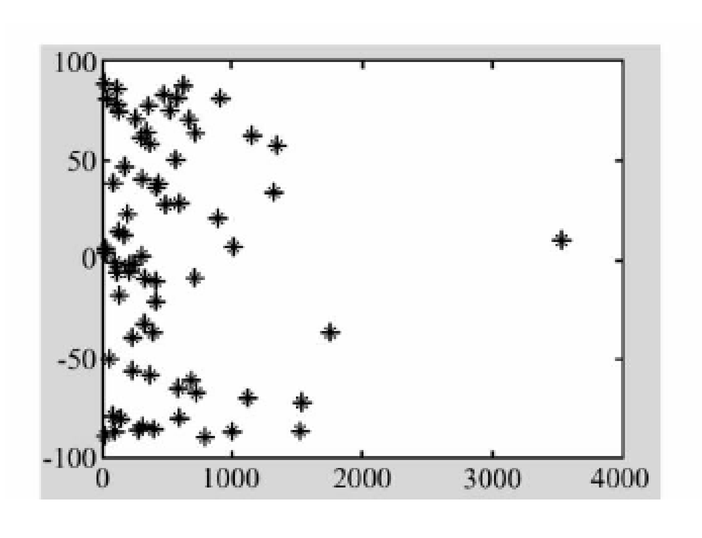


图 9.25　特征图

【例 9.7】统计某一大小的星体所占的比例。

第一步将图像读入。

```
I=imread('ngc4024l.tif');
subplot(1,2,1);
imshow(I);title('原始图像');
```

第二步为了充分利用 MATLA 中向量语言处理的功能,我们将 unit8 型数据图像转换为 double 型数据图像;为了拉伸图像的对比度,用 imadjust 进行灰度扩展,将图像的灰度扩展到整个灰度范围[0 1]。

```
gI=imadjust(im2double(I), [], [0 1]);
subplot(1,2,2);
Imshow(gI), title('灰度调整后的图像');
```

其显示结果如图 9.26 所示。

第三步观察灰度调整后的图像,可以看到,在图像的中心位置,由于背景较亮,所以前景显示很模糊。由于对象的灰度比背景的灰度亮,采用高帽变换消除图像背景中那些不一致的灰度数。这里采用一个尺寸较大的结构元素来进行高帽变换。

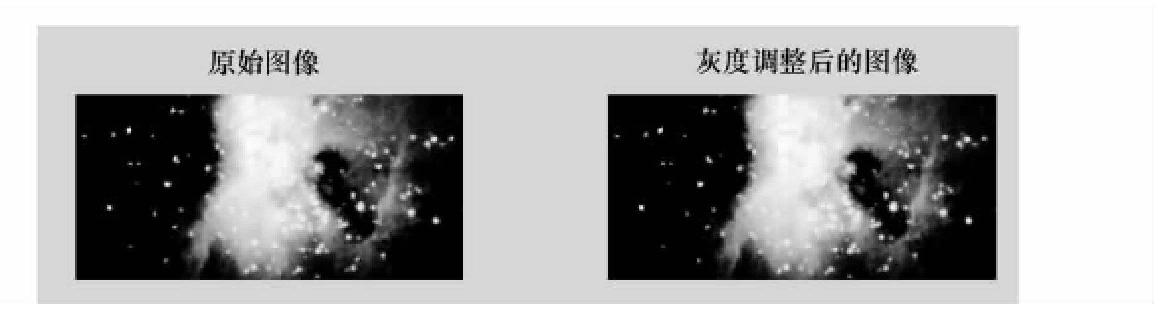


图 9.26　原始图像和对比度增强图

```
se=strel('disk', 10);
topI=imtophat(gI, se);
figure, imshow(topI), title('高帽变换结果');
```

其显示结果如图 9.27 所示。

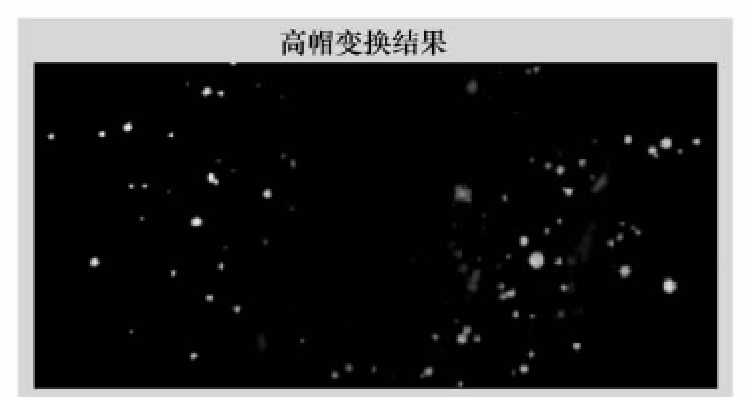


图 9.27　高帽变换后图像

第四步对高帽变换后的图像采用粒度测试技术来分析原始图像中星体大小的分布情况。这里可以利用一个逐渐变大的结构元素来不断地进行图像形态开启，并统计开启后图像的剩余面积，通过绘制结构元素的大小和剩余面积的大小就可以计算出各种大小的星体在图像中占有的比例。这里采用一个 for 循环来完成：

```
for counter=0:20
    remain=imopen(topI, strel('disk', counter));
    surfarea(counter + 1) = sum(remain(:));
end
figure, plot(surfarea, 'm - * '), grid on;
set(gca, 'xtick', [0 2 4 6 8 10 12 14 16 18 20]);
```

其显示结果如图 9.28 所示。

第五步从曲线图可以看出，随着结构元素的增大，对象的剩余面积发生锐减。这是由于原始图像中含有较多的大小相同的星体的缘故。通过计算两次开启操作前、后的斜率(即一阶偏导)就可以估计出图像中大小相同的星体所占的比例。

```
derivsurfarea=diff(surfarea);
figure, plot(derivsurfarea, 'm- * '), grid on;
```

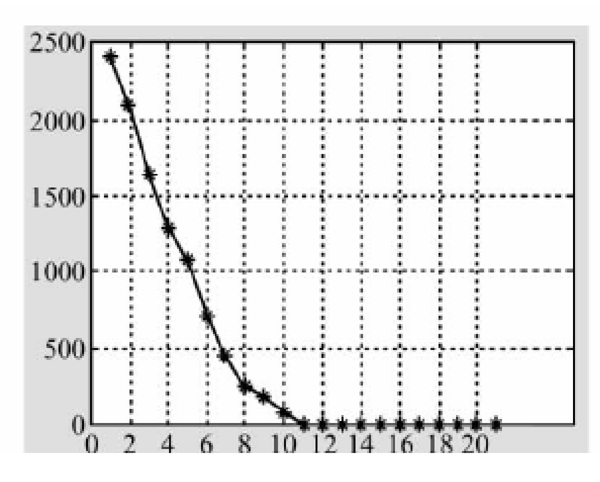


图 9.28 统计图

其显示结果如图 9.29 所示。

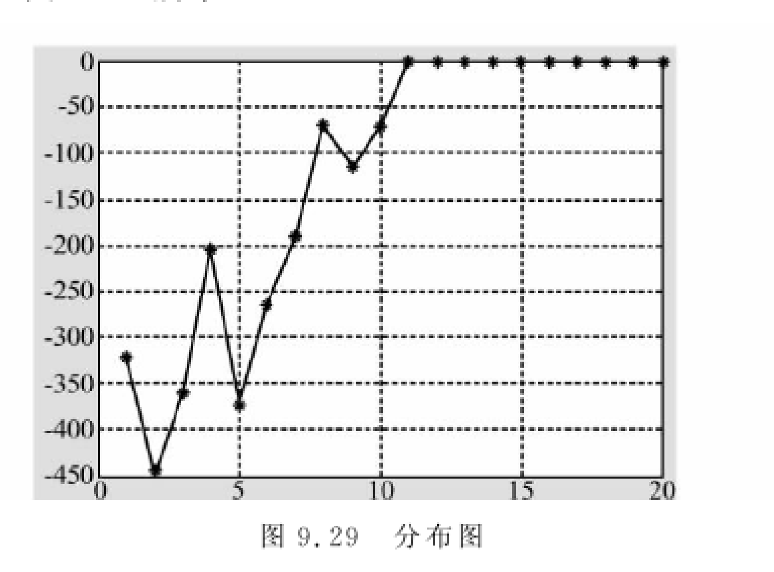


图 9.29 分布图

从最后的结果可以看出星体大小分布情况，半径为 2 个像素的星体大约占了 450 个像素，半径为 5 个像素的星体大约占了 375 个像素，半径为 9 个像素的星体大约占了 125 个像素。

9.4 图像去噪

【例 9.8】给定一个只含有较少噪声的图像 facets.mat，试用二维小波分析进行图像去噪处理。

分析：对于有些图像的噪声，由于只是含有少量的高频噪声，如果采用把高频噪声全部滤除的方法将会损害图像中固有的高频有用信号，使图像的细节受到很大损害。这种情况下可以采用高频系数阈值量化的方法进行处理。这里采用小波分解后，将高频细节分量进行阈值量化。

```
%装入图像
load facets;
```

```
%显示原始图像
subplot(1,3,1);image(X);colormap(map);
title('原始图像');
axis square;
%产生噪声
init=2055615888; randn('seed',init);
x=X+15 * randn(size(X));
%显示含噪声图像
subplot(1,3,2);image(x);colormap(map);
title('含噪声图像');
axis square;
%以下代码将进行去噪处理
%用 coif3 小波基对 x 进行二层小波分解
[c,s]=wavedec2(x,2,'coif3');
%设置尺度向量 n,用于说明要处理的细节分量的层次
n=[1,2];
%设置阈值向量 p,对每层细节分量设定处理的阈值
p=[10.25,23.38];
%对三个高频细节分量进行阈值处理
nc=wthcoef2('h',c,s,n,p,'s');
nc=wthcoef2('v',c,s,n,p,'s');
nc=wthcoef2('d',c,s,n,p,'s');
%对新的小波分解结构[nc,s]重构
xx= waverec2(nc,s,'coif3');
%显示去噪后的图像
subplot(1,3,3);image(xx)
title('去除噪声后的图像');
axis square;
```

其显示结果如图 9.30 所示。

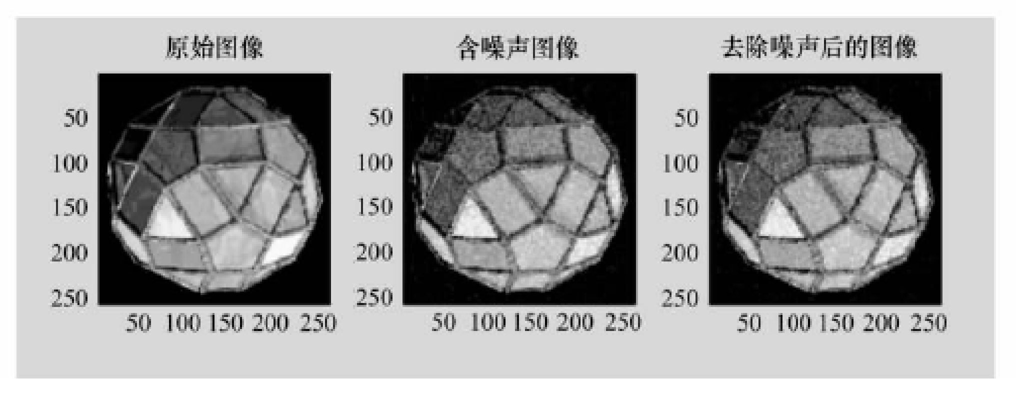


图 9.30　小波变换对图像压缩

第 10 章　Matlab GUI 设计

10.1　图形用户界面简介

计算机用户界面是指计算机与其使用者之间的对话接口,是计算机系统的重要组成部分。计算机的发展史不仅是计算机本身处理速度和存储容量飞速提高的历史,而且是计算用户界面不断改进的历史。早期的计算机是通过面板上的指示灯来显示二进制数据和指令,人们则通过面板上的开关、扳键及穿孔纸带送入各种数据和命令。20 世纪 50 年代中、后期,由于采用了作业控制语言(Job Control Language,JCL)及控制台打字机等,使计算机可以批处理多个计算任务,从而代替了原来笨拙的手工扳键方式,提高了计算机的使用效率。

20 世纪 80 年代初,由美国 Xerox 公司 Alto 计算机首先使用的 Smalltalk-80 程序设计开发环境以及后来的 Lisa、Macintosh 等计算机,将用户界面推向图形用户界面(Graphical User Interface,GUI)的新阶段。随之而来的用户界面管理系统和智能界面的研究均推动了用户界面的发展。用户界面已经从过去的人去适应笨拙的计算机,发展到今天的计算机不断适应人的需求。

用户界面的重要性在于它极大地影响了最终用户的使用,影响了计算机的推广应用,甚至影响了人们的工作和生活。由于开发用户界面的工作量极大,加上不同用户对界面的要求也不尽相同,因此,用户界面已成为计算机软件研制中最困难的部分之一。当前,Internet 的发展异常迅猛,虚拟现实、科学计算可视化及多媒体技术等对用户界面提出了更高的要求。

图形用户界面的广泛流行是当今计算机技术的重大成就之一,它极大地方便了非专业用户的使用,人们不再需要死记硬背大量的命令,而可以通过窗口、菜单方便地进行操作。归纳起来,图形用户界面的主要特征有以下几个。

(1) WIMP。W(Windows)指窗口,是用户或系统的一个工作区域。一个屏幕上可以有多个窗口。I(Icons)指图符,是形象化的图形标志,易于人们隐喻和理解。M(Menu)指菜单,是可供用户选择的功能提示。P(Pointing Devices) 指鼠标器等,便于用户直接对屏幕对象进行操作。

(2) 用户模型。GUI 采用了不少桌面办公的隐喻,使应用者共享一个直观的界面框架。由于人们熟悉办公桌的情况,因而容易理解计算机显示的图符的含义,

如文件夹、收件箱、画笔、工作簿、钥匙及时钟等。

(3) 直接操作。过去的界面不仅需要记忆大量命令,而且需要指定操作对象的位置,如行号、空格数、*X* 及 *Y* 的坐标等。采用 GUI 后,用户可直接对屏幕上的对象进行操作,如拖动、删除、插入以及放大和旋转等。用户执行操作后,屏幕能立即给出反馈信息或结果,因而称为所见即所得(What You See Is What You Get)。用视、点(鼠标)代替了记、击(键盘),给用户带来了方便。

本章介绍图形用户界面的功能,这使用户能够定制自己与 Matlab 的交互方式,命令窗口不再是唯一与 Matlab 的交互方式。其中将详细说明图形句柄(标识符)uicontrol 和 uimenu 对象的使用,把图形界面加到 Matlab 的函数和 M 文件。uimenu 对象能在图形窗口中产生下拉式菜单和子菜单。uicontrol 对象能建立如按钮、滚动条、弹出式菜单以及文本框等对象,最后将结合一个实例来说明如何更好地使用 Matlab GUI 编程。

10.2 Matlab 图形对象介绍

Matlab 图形对象是构建 Matlab GUI 的基础,每一个 Matlab 图形对象都和一个唯一的句柄相关联,通过句柄用户就可以对图形对象的属性进行访问和修改,在创建一个图形对象时系统返回的也是这样一个句柄。

Matlab 图形库的所有对象被组织成一个树状结构(见图 10.1)。

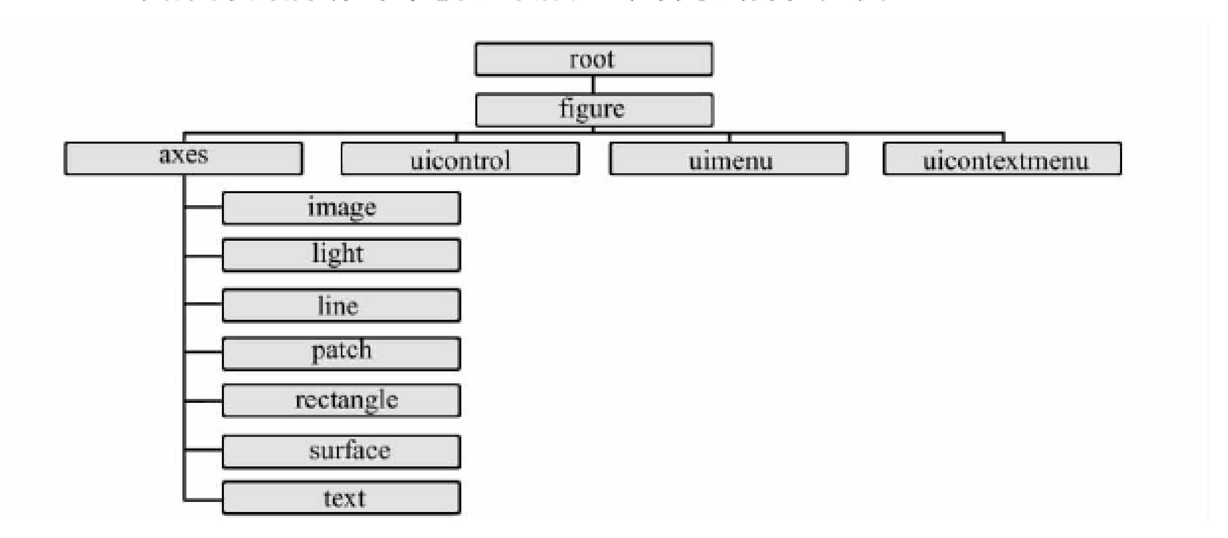


图 10.1 Matlab GUI 对象树状结构图

10.2.1 axes 对象

有了这张树形图,就可以据此创建自己所需的对象,图 10.2 所示的 Matlab 图形窗口中包括了 figure、surface、axes(3D)、axes(2D)、image、line 等对象,例如,需要在 axes 中创建一个 text 对象时,可以使用下面的代码:

```
text(x,y,'string')
```

```
text(x,y,z,'string')
text(...'PropertyName',PropertyValue...)
h = text(...)
```

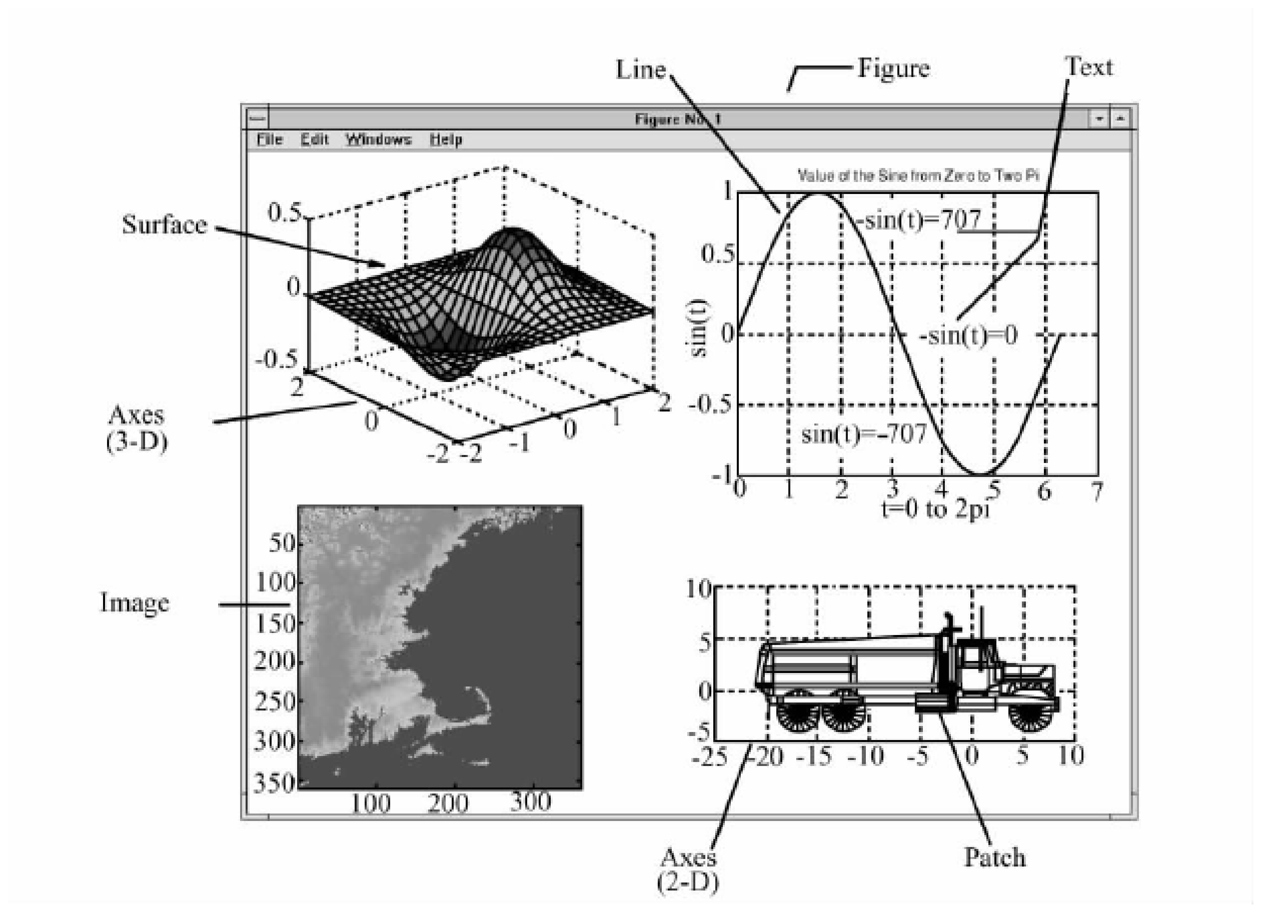


图 10.2 坐标轴对象示例

实际上所有的对象创建时都有着相同的格式：

```
handle = function('propertyname',propertyvalue,...)
```

Matlab GUI 对象及其描述见表 10.1。

表 10.1 Matlab GUI 对象描述列表

函数	对象描述
axes	坐标轴对象，可以包含 image、light、line、patch、surface、text 等子对象
figure	显示图形的窗口
image	2D 图形，可以表示 8 位或双精度的图像
light	光源对象
line	线条对象
rectangle	2D 填充区域，可以是从矩形到椭圆的多种图形
surface	3D 区域中用于表示图像表面的对象
text	图形中的文本对象
uicontextmenu	与图形对象相关联的上下文菜单对象
uicontrol	可编程的图形控件，如 pushbutton、slider、listbox
uimenu	图形窗口的主菜单

图形的构造函数返回的都是对象的句柄，创建时用户还可以指定对象的属性，这些属性值都是以 property/value 对的方式传递给构造函数的。同样，用户可以以这种方式修改和查询对象的任何属性值。

下例中 surface 函数并没有显示一个三维的图形，因为 surface 对象的创建函数并没有改变 axes 对象的属性（除了 children 属性外）。

```
[x,y]= meshgrid([-2:.4:2]);
Z=x.*exp(-x.^2-y.^2);
fh=figure('Position',[350 275 400 300],'Color','w');
ah=axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],
        'YTick',[-2 -1 0 1 2]);
sh=surface('XData',x,'YData',y,'ZData',Z,
           'FaceColor',get(ah,'Color')+.1,
           'EdgeColor','k','Marker','o',
           'MarkerFaceColor',[.5 1 .85]);
```

这段程序在 Matlab 中执行后的显示结果如图 10.3 所示。

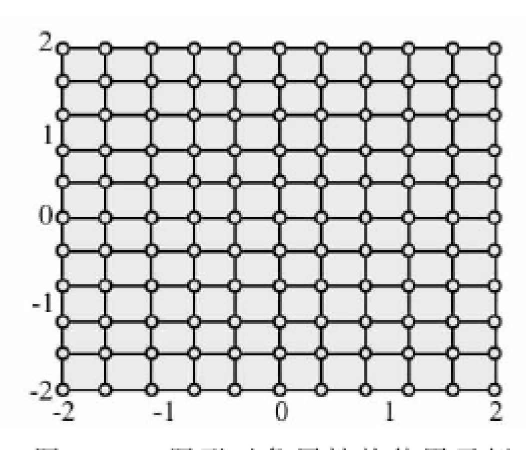


图 10.3 图形对象属性的使用示例

如果需要显示三维图像，必须使用 camera 函数或者直接用下面的指令：

```
view(3)
```

其显示结果如图 10.4 所示。

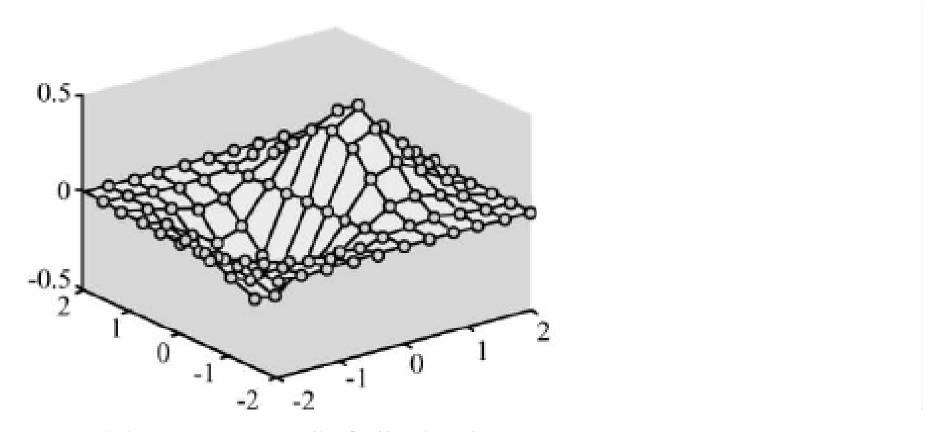


图 10.4 view 指令使用示例

在通常情况下，使用创建函数生成的图形对象都有默认的所属对象，例如，创建一个坐标轴对象，它将默认地隶属于当前的图形窗口，一个线条对象隶属于当前的坐标轴，Matlab 图形库也提供了接口让用户改变所属关系，如下面的语句：

```
axes('Parent',figure_handle,...)
```

这条语句既指定了坐标轴创建于哪个图形窗口，同时也可以改变一个对象的所属关系，例如：

```
set(gca,'Parent',figure_handle)
```

图形对象的创建函数还提供了简化调用的功能，如下面的两个语句是等价的：

```
text(.5,.5,.5,'Hello')
text('Position',[.5 .5 .5],'String','Hello')
```

其中省略的是属性的名称，因此使用时必须注意不能出现冲突的情况。

Matlab 图形库提供的操作对象属性的方法是 get 和 set，如下面的语句：

```
set(object_handle,'PropertyName','NewPropertyValue');
returned_value=get(object_handle,'PropertyName');
```

这种方法就是以 property/value 对的方式访问对象属性，用户还可以使用 findobj 来查询对象的信息，下面是调用语法：

```
h=findobj
h=findobj('PropertyName',PropertyValue,...)
h=findobj(objhandles,...)
h=findobj(objhandles,'flat','PropertyName',PropertyValue,...)
```

例如，当需要查找一个坐标轴中的所有线条对象时，就可以使用下面的语句：

```
h=findobj(gca,'Type','line')
```

set 方法还支持模糊查询，下面这条 set 语句列出了 Marker 属性的所有可能值：

```
set(obj_handle,'Marker')
```

返回的结果如下（括号中的为默认值）：

```
[+|o|*|.|x|square|diamond|v|^|>|<|pentagram
|hexagram |{none}]
```

set 指令还可以列出一个对象的所有可以修改的属性，如下面的指令列出了 axes 对象所有可以修改的属性：

```
set(axes);
ALim
    ALimMode: [{auto}| manual]
    AmbientLightColor
    Box: [on| {off}]
    CameraPosition
    CameraPositionMode: [{auto}|manual]
```

CameraTarget
……
……

ZLim
ZLimMode：[{auto}|manual]
ZMinorGrid：[on|{off}]
ZMinorTick：[on|{off}]
ZScale：[{linear}|log]
ZTick
ZTickLabel
ZTickLabelMode：[{auto}|manual]
ZTickMode：[{auto}|manual]
ButtonDownFcn：string -or- function handle -or- cell array
Children
Clipping：[{on}|off]
CreateFcn：string-or-function handle-or-cell array
DeleteFcn：string-or-function handle -or-cell array
BusyAction：[{queue}|cancel]
HandleVisibility：[{on}|callback|off]
HitTest：[{on}|off]
Interruptible：[{on}|off]
Parent
Selected：[on|off]
SelectionHighlight：[{on}|off]
Tag
UIContextMenu
UserData
Visible：[{on}|off]

其中，花括号中的都是默认的属性值。

对于这些默认值，实际上用户也可以自己设置。设置的时候调用 set 指令，其中属性名称的参数以 Default 开头，然后跟上对象名称，最后跟上所需要的属性名，这样就可以设置默认的属性值。如：

```
set(gcf,'DefaultLineLineWidth',1.5)
```

这条语句对当前图形窗口和当前坐标轴的 Line 对象的 LineWidth 属性设置一个 1.5 的默认值。如果需要对不同的级别的属性更改默认值，需要将 set 函数的第一个参数设定为所需要的级别，例如，下面的语句更改了所有级别的图形的背景色的默认值。

```
set(0,'DefaultFigureColor','b')
```

下面是两个复杂一些的实例，第一个例子中创建一个以白色作为背景色的坐标轴，然后从根级别设置 axes 对象所有的属性的默认值。

```
whitebg('w')
set(0,'DefaultAxesColorOrder',[0 0 0],
  'DefaultAxesLineStyleOrder','-|--|:|-.')
```

然后调用 plot 函数：

```
Z=peaks; plot(1:49,Z(4:7,:))
```

这段程序在 Matlab 中执行后的显示结果如图 10.5 所示。

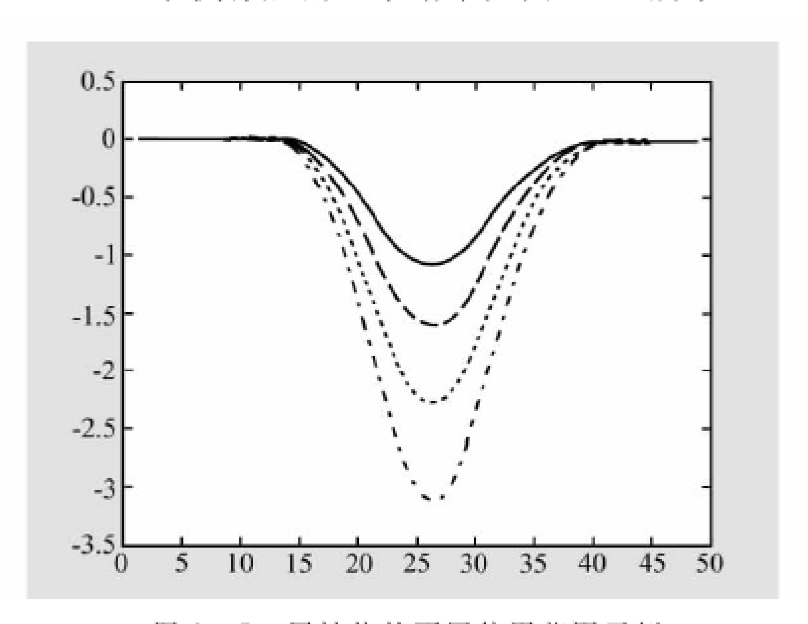


图 10.5　属性值的不同使用范围示例

可以看到坐标轴中的所有点使用的都是 set 语句中指定的默认的单色，而线条的类型也是 set 函数中指定的几种类型之一。

第二个例子中增加了对默认值级别的控制，下面的代码将在一个图形窗口中创建两个坐标轴，并对 figure 和 axes 分别做不同的级别控制：

```
t=0:pi/20:2*pi;
s=sin(t);
c=cos(t);
% 设置 axes 对象的 Color 属性默认值
figh=figure('Position',[30 100 800 350],
          'DefaultAxesColor',[.8 .8 .8]);

axh1=subplot(1,2,1); grid on
% 为第一个坐标轴设置 LineStyle 值
set(axh1,'DefaultLineLineStyle','-.')
line('XData',t,' YD ata',s)
line('XData',t,' YD ata',c)
```

```
text('Position',[3 .4],'String','Sine')
text('Position',[2 -.3],'String','Cosine',
    'HorizontalAlignment','right')

axh2=subplot(1,2,2); grid on
% 设置第二个坐标轴的 text Rotation 属性
set(axh2,'DefaultTextRotation',90)
line('XData',t,' YD ata',s)
line('XData',t,' YD ata',c)
text('Position',[3 .4],'String','Sine')
text('Position',[2 -.3],'String','Cosine',
    'HorizontalAlignment','right')
```

这段程序在 Matlab 中执行后的显示结果如图 10.6 所示。

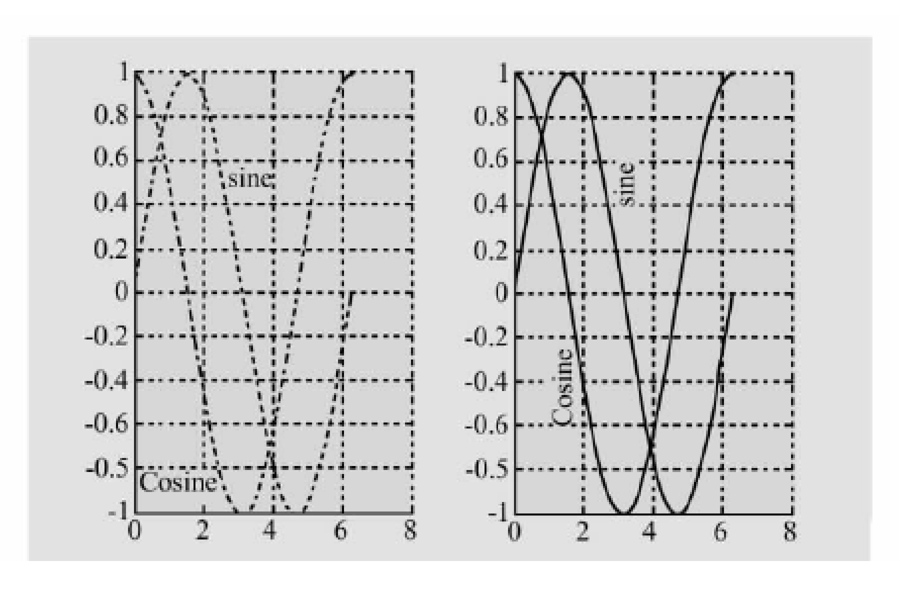


图 10.6　对默认值级别的控制示例

get 函数最简单也最常用的用法是下面的语句：

```
get(gca,'PlotBoxAspectRatio')
ans =
  1      1      1
```

这个指令得到当前坐标轴的比例刻度。

用户可以使用 get 指令来得到一个对象所有的属性，但是对于那些包含数据的属性，一般来说 Matlab 都只会给出它们的维度，如下面的 CurrentPoint 和 ColorOrder 属性：

```
get(gca);
ALim=[0 1]
      ALimMode=auto
      AmbientLightColor=[1 1 1]
```

```
Box=off
CameraPosition=[0.5 0.5 9.16025]
CameraPositionMode=auto
CameraTarget=[0.5 0.5 0.5]
CameraTargetMode=auto
CameraUpVector=[0 1 0]
CameraUpVectorMode=auto
CameraViewAngle=[6.60861]
CameraViewAngleMode=auto
CLim=[0 1]
CLimMode=auto
Color=[1 1 1]
CurrentPoint=[ (2 by 3) double array]
ColorOrder=[ (7 by 3) double array]
……
ZLabel=[104]
ZLim=[0 1]
ZLimMode=auto
ZMinorGrid=off
ZMinorTick=off
ZScale=linear
ZTick=[0 0.5 1]
ZTickLabel=
ZTickLabelMode=auto
ZTickMode=auto

BeingDeleted=off
ButtonDownFcn=
……
Visible=on
```

get 指令还可以对一组属性进行查询，例如，下面的例子中先设置一个属性数组：

```
camera_props(1) = {'CameraPositionMode'};
camera_props(2) = {'CameraTargetMode'};
camera_props(3) = {'CameraUpVectorMode'};
camera_props(4) = {'CameraViewAngleMode'};
```

然后调用 get 指令进行查询，得到的结果也是一个数组：

```
get(gca,camera_props)
```

```
ans=
  'auto' 'auto' 'auto' 'auto'
```

Matlab 还提供了 copyobj 函数来将一个图形对象复制到新的空间中，新对象与原对象不同之处在于其 parent 属性和对象句柄发生了变化，其他的部分都是相同的，下例中使用了 text 函数来指定一个坐标点的信息：

```
text('String','\{5\pi\div4, sin(5\pi\div4)\}\rightarrow',
  'Position',[5 * pi/4,sin(5 * pi/4),0],...
  'HorizontalAlignment','right')
```

注意，这个字符串参数使用了 text 指令，除了输出文本外，还输出一个向右的箭头，同时 Position 参数指定了这个 text 对象出现的位置，HorizontalAlignment 指定这个对象的对齐方式。结果如图 10.7 所示。

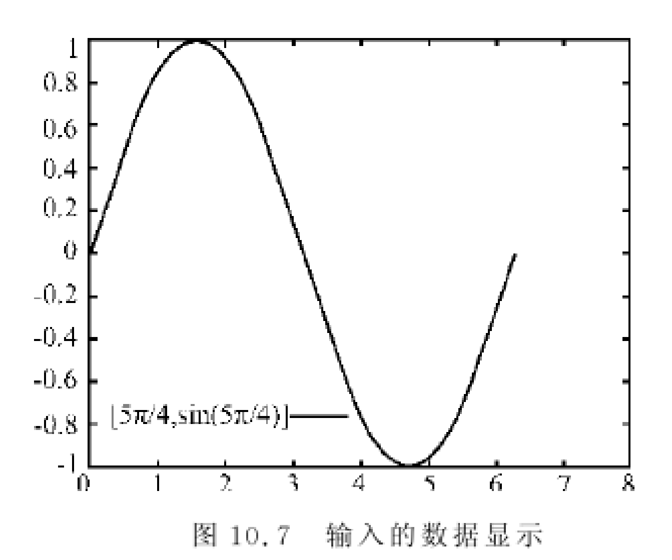


图 10.7　输入的数据显示

```
text_handle = findobj('String','\{5\pi\div4,sin(5\pi\div4)\}\rightarrow');
copyobj(text_handle,gca)
```

当创建新的图像时就可以通过这个复制的对象显示文本信息了，如图 10.8 所示。

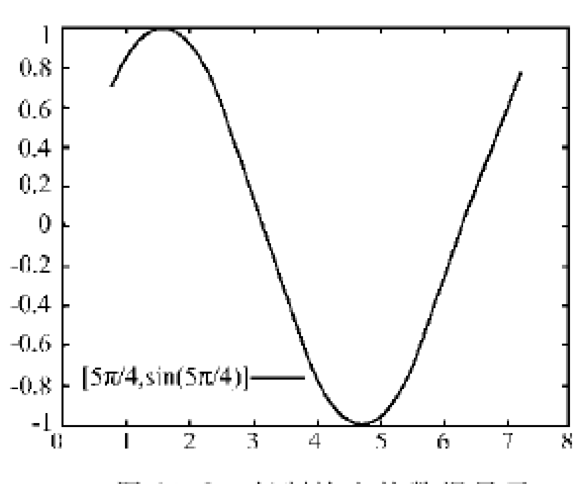


图 10.8　复制输出的数据显示

10.2.2 uimenu 对象

uimenu 对象代表了图形窗口的标准菜单，通常显示在窗口的顶部。一个菜单项还可用自己的菜单项列表作为子菜单。子菜单项在子菜单的标志右边显示小三角或箭头以表示该菜单还有更多子菜单项可供选择。如果子菜单的菜单项被选择，另一个具有更多菜单项的菜单显示在此菜单的右边的下拉菜单中。有时这种菜单称为行走菜单，选中其中一个菜单项也引起某些动作的产生。

子菜单可以嵌套，但嵌套层次的数目受窗口系统及有用资源的限制。

下面这段代码显示了如何控制菜单对象 uimenu 的显示。

```
%获得默认设置的标准菜单
figure;


% 如何隐去标准菜单
set(H_fig ,'MenuBar','none');
set(gcf,'menubar',menubar);


% 恢复图形窗上标准菜单
set(gcf,'menubar','figure');
```

Matlab 中使用函数 uimenu 建立菜单项。uimenu 的句法与其他对象创建函数相似。例如：

```
>> Handle_1=uimenu(Hx_parent,'PropertyName' ,PropertyValue,...)
```

其中 Handle_1 是由 uimenu 生成的菜单项的句柄，通过设定 uimenu 对象的属性值“'PropertyName' ,PropertyValue”这对命令定义了菜单特性；Hx_parent 是默认的父辈对象的句柄，必须是图形和 uimenu 对象。

uimenu 对象中最重要的属性是“Label”和“Callback”。“Label”属性值是菜单条和下拉菜单项上的文本字符串，以确认菜单项。“Callback”属性值是 Matlab 字符串，当选中菜单项时，它传给 eval，用以执行。

下例用函数 uimenu 将简单菜单加到当前的图形窗口中。这里提出的例子说明如何只用几个 Matlab 命令来建立工作菜单。后面的例子将详细讨论 uimenu 的命令和属性。

下例用两个下拉菜单将菜单条加到当前窗口中。首先，建立名为 Example 的顶部菜单输入：

```
>> Handle_ex=uimenu(gcf,'Label ','Example');
```

在此菜单下有两个菜单项。第一项标志为 Grid，切换坐标轴格栅的状态：

```
>> Handle_exgrid=uimenu(Handle_ex,'Label ','Grid ','Callback' ,'Grid');
```

注意，句柄 Handle_ex 是用于与上层菜单相关联的。这项 uimenu 按 eval 的

要求出现在上层菜单之下。还需注意的是属性“Callback”的值，它是一个带引号的字符串。

Example 下的第二项标志为 View，并带有子菜单：

```
>> Handle_exview=uimenu(Handle_ex,'Label','View');
```

View 菜单有两项选择“2－D”和“3－D”视图：

```
>> Handle_ex2d=uimenu(Handle_exview,'Label ','2－D ','Callback ','view(2)
  ');
>> Handle_ex3d=uimenu(Handle_exview,'Label ','3－D ',' Callback' ,'View(3)'
  );
```

注意，以上这些是 View 的子菜单，因为它们指定 Handle_exview 作为其父对象。

现在，将第二个顶层菜单加到标题为 Close 的菜单条中：

```
>> Handle_ex=uimenu(gcf, ' Label ' , ' Close ' );
```

由该顶层菜单 Close 加入了两个菜单项。第一项关闭图形窗口，第二项使图形窗口打开，但去掉用户菜单：

```
>> Handle__clfig = uimenu ( Handle__close,' Label ',' Close Figure',' Callback ','
Close ');
>> Handle_clmenu=uimenu(Handle_close,'Label','Remove Menu ',...
                        ' Callback',' delete ( Handle __ ex ); delete ( Handle __ ex );
drawnow' );
```

如上所列，同句柄图形函数一样，在建立图形对象时可定义 uimenu 属性，或用 set 改变属性。所有可设定的属性，包括标题、菜单颜色，甚至回调字符串都可以用 set 来改变。这种功能十分便于迅速地定制菜单和属性。

表 10.2 列出了 Matlab 中的 uimenu 对象的属性及其属性值。带有“ * ”的属性是非文件式的，使用时需加小心。在“{}”内的属性值是默认值。

表 10.2　Uimenu 对象的属性列表

属 性	说　明
Accelerator	指定菜单项等价的按键或快捷键。对于 X-Windows 系统，按键顺序是“Control-字符”；对于 Macintosh 系统，按键顺序是“Command-字符”或“# -字符”
BackgroundColor	uimenu 背景色，是一个 3 元素的 RGB 向量或 Matlab 预先定义的颜色名称。默认的背景色是亮灰色
Callback	Matlab 回调字符串，选择菜单项时，回调字符串传给函数 eval。初始值为空矩阵
Checked	被选项的校验标记
on：	校验标记出现在所选项的旁边
{off}：	校验标记不显示

Enable	菜单使能状态
〈on〉：	菜单项使能。选择菜单项能将 Callback 字符串传给 eval

（续）

属 性	说 明
off：	菜单项不使能，菜单标志变灰。选择菜单项不起任何作用。
ForegroundColor	uimenu 前景(文本)色，是一个 3 元素的 RGB 向量或 Matlab 预先定义的颜色名称。默认的前景色是黑色
Label	含有菜单项标志的文本串。在 PC 系统中，标记中前面有“&”，定义了快捷键，由“Alt- 字符“激活
Position	uimenu 对象的相对位置。顶层菜单从左到右编号，子菜单从上至下编号
Separator	分割符—线模式
on：	分割线在菜单项之上
〈off〉：	不画分割线
* Visible	uimenu 对象的可视性
〈on〉：	uimenu 对象在屏幕上可见
off：	uimenu 对象不可见
ButtonDownFcn	当对象被选择时，Matlab 的回调字符串传给函数 eval。初始值为空矩阵
Children	其他 uimenu 对象的句柄
Clipping	限幅模式
〈on〉：	对 uimenu 对象有效果
off：	对 uimenu 对象无效果
DestroyFcn	没有文本说明(仅用于 Macintosh 4.2 版本)
Interrruptible	指明 ButtonDownFcn 和 CallBack 串可否中断
〈no〉：	回调串不可中断
yes：	回调串可中断
Parent	父对象的句柄；如果 uimenu 对象是顶层菜单，则为图形对象；若 uimenu 是子菜单，则为父的 uimenu 对象句柄
* Select	值为[on|off]
* Tag	文本串
Type	只读对象辨识串，通常为 uimenu
UserData	用户指定的数据。可以是矩阵，字符串等

属性值定义了 uimenu 对象的性质并且控制菜单如何显示，也决定了选择菜单项所引起的动作。下面更详细地讨论其中某些性质。

“Label”属性定义了出现在菜单或菜单项中的标志。它也可以用来定义 Microsoft Windows 系统的快捷键：标志字符串中，在所需字符前加上“&”，例如：

```
>> Handle_top=uimenu('Label','Example');
>> uimenu(Handle_top,'Label','&Grid','CallBack','grid');
```

它定义了键盘上 G 为快捷键。菜单项标志将以 Grid 形式出现在菜单上。为激活快捷键，在选择图形窗口时按 Alt 键并按下 G 键。快捷键不一定是字符串的第一字符。下例中 R 为快捷键：

```
>> uimenu(Handle_top,'Label','G & rid','CallBack','grid');
```

则标志以 Grid 形式出现在菜单上。Macintosh 平台用" Accelerator"属性而不是" Label"来定义快捷键。在 Macintosh 中，使用下面的代码定义 G 为快捷键：

```
>> uimenu(Handle_top,'Label','Grid','Accelerator' ,'G','CallBack',' grid');
```

其中菜单标志以"Grid ♯ G"的形式出现。为激活快捷键，选择图形窗口时，按 Command 键或 ♯ 键并按下 G 键。

影响菜单的布置和外观的三个属性为 Position、Checked 和 Separator。uimenu 对象的 Position 属性值是一个整数，它定义了相对于其他菜单和菜单项的位置。在生成菜单时，设定 Position 属性。菜单条最左端的菜单条和下拉菜单中的上端菜单项处在位置 1。

设置 Position 属性可以重新排列菜单位置。例如：

```
>> Handle_1=uimenu('Label' ,'first');      % Create two menus
>> Handle_2=uimenu('Label','Second');
>> get(Handle_1,'Position')               % Check the locations
ans=
     1

>> get(Handle_2,'Position')
ans=
     2

>> set(Handle_2,'Position',1)             % Change menu order
>> get(Handle_1,'Position')               % check the locations

ans=
     2

>> get(Handle_2,'Position')

ans=
     1
```

注意，当一个 uimenu 的 Position 属性改变，其他 uimenu 的 Position 属性均更

新以适应此变化。子菜单中菜单选项的编号以同样的方式重新排列。属性 Checked 的值使校验标记出现在菜单项标志的左边。默认值为 off。

```
>> set(Handle_item,'Checked','on')
```

上面命令使校验标记出现在 Handle_item uimenu 标志的旁边。对于创建代表属性的菜单项,该命令十分有用。例如:

```
>>Handle_top=uimenu('Label','Example');
>>Handle_box=uimenu(Handle_top,'Label','Axis Box',...
                    'CallBack',[...
                    'if strcmp(get(fca,''Box'' ),''on''),' ,...
                    'set(gca,''Box'',''off''),',...
                    set(Handle_box,''Checked'',...off''),',...
                    'else,',...
                    'set(gca,''Box'',''on''),',...
                    'set(Handle_box,''Checked'',''on''),',...'end']);
```

上述代码建立了以 Axis Box 为标志的下拉菜单项,当选中该项时,就运行回调字符串所表示的命令。回调字符串确定了当前坐标轴的 Box 属性值,并适当地设定坐标轴的 Box 属性及 uimenu 的 Checked 属性。该例可从精通 Matlab 工具箱的 M 脚本文件 mmenu2.m 中得到。

可以通过改变 uimenu 的 Label 属性反映菜单项当前的状态。下例(在 mmenu3.m 中)改变了菜单项标志本身,而不是加一个校验标记:

```
>> Handle_top=uimenu('Label','Example');


>> Handle_box+uimenu(Handle_top,'Label','Axis Box',...
                    'CallBack',[...
                    'if strcmp(get(gca,''Box''),''on''),' ,...
                    'set(gca,''Box'',''off''),',...
                    set(Handle_box,''Label'',''Set Box On''),',...
                    'else,',...
                    'set(gca,''Box'',''on'' ),' ,...
                    'set(Handle_box,''Label'',''Set box Off''),',...
                    'end']);
```

使用 Separator 属性可将下拉菜单分成若干逻辑组。如果一个 uimenu 项的 Separator 属性是"on",则在下拉菜单中显示此项时,此项的上端有一条水平线将其与前面的菜单项隔开。默认值是"off",但在菜单生成时可以改变。例如:

```
>> Handle_box=uimenu(Handle_top,'Label','Box','Seperator','on');
```

或是在以后使用 set 命令:

```
>> set(Handle_box,'Separator','on');
```

顶层菜单忽略 Seprator 的属性值。

下面这段代码把用户菜单 Option 设置为顶层的第三菜单项,同时下拉菜单被两条分隔线分为三个菜单区,而最下面的菜单项又由两个子菜单组成。

```
% uimenu example
figure
h_menu=uimenu('label','Option','Position',3);
h_sub1=uimenu(h_menu,'label','grid on','callback','grid on');
h_sub2=uimenu(h_menu,'label','grid off','callback','grid off');
h_sub3=uimenu(h_menu,'label','box on','callback','box on',...
    'separator','on');

h_sub4=uimenu(h_menu,'label','box off','callback','box off');
h_sub5=uimenu(h_menu,'label','Figure Color','Separator','on');
h_subsub1=uimenu(h_sub5,'label','Red','ForeGroundColor','r',...
    'callback','set(gcf,''Color'',''r'')');
h_subsub2=uimenu(h_sub5,'label','Reset',...
    'callback','set(gcf,''Color'',''w'')');
```

这段程序在 Matlab 中执行后的显示结果如图 10.9 所示。

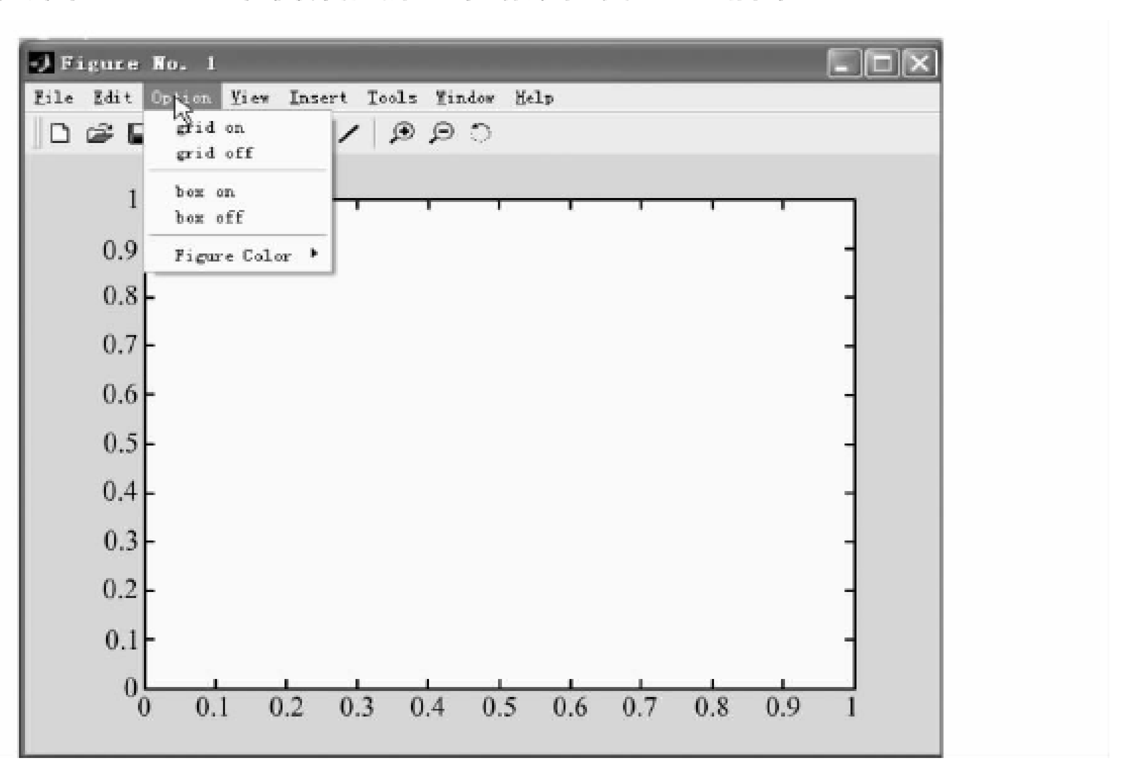


图 10.9　Matlab 菜单对象示例

图 10.9 中显示了加入的 option 菜单包括五个子菜单项,其中 Figure Color 菜单项还包括下一层的菜单项,而响应菜单事件的动作同样在 uimenu 函数中以 callback 的形式指定。Callback 属性的值是一个命令字符串,Matlab 将它传给函

数 eval 并在命令窗口工作空间执行。

因为 Callback 属性必须是字符串，所以在字符号内多重 Matlab 命令、后续行以及字符串都会使必需的句法变得十分复杂。如果有不止一个命令要执行，命令间必须适当地分隔开。例如：

```
>>uimenu(' Label','Test','CallBack','grid on; set(gca,' 'Box' ',' 'on' ')');
```

把一个字符串传给 eval，使用命令：

```
>> grid on; set(gca,'Box','on')
```

此命令在命令窗口工作空间中执行。这是合法的句法，因为命令用逗号或分号隔开，多重命令可输入到同一命令行中。在定义回调函数时，也遵循 Matlab 规定，即在已引用的字符串内，用两个单引号来表示单引号。

字符串可以串接起来生成一个合法 Matlab 字符串，只要把它们括在方括号中。例如：

```
>>uimenu('Label','Test','CallBack',['grid on,',' set(gca,' 'Box' ',' 'on' ' )']);
```

注意，字符串“grid on”含有所需的逗号以分隔两个命令。

如果使用了续行号，上述命令可写为：

```
>>uimenu('Label','Test',...
          'CallBack',[...
          'grid on,',...
          ' set(gca,' ' Box' ',' 'on' ' )' ...
          ]);
```

上例中命令行被分隔，每行的末尾加上了三个句点表示命令的继续。上列单行的所有元素都被保留，包括字符串分隔命令的逗号。在“grid on,...” 行中最后引号后的逗号是可选的；下一行开始的空格起相同的作用。

由于在命令字符串中使用标点符号比较复杂，而且不易查错，因此使用时一般遵照以下规则：

(1) 把整个回调字符串括在方括号中，不要忘记最后的右括号“)”；

(2) 把各语句括上单引号；

(3) 已引用的字符串内，要用双引号，如 ' quoted'：'a ' 'quoted' ' string';' Quote ' 'a ' ' ' 'quoted' ' ' 'string''now'，在引号后要用逗号或空格结尾；

(4)除了最后一句，各语句在引号内要以逗号或分号结尾，在引号后要用逗号或空格结尾；

(5) 有后续行的各行要以三个句点(“...”)结尾。

再如：

```
>> Hm_top= unimenu(Label','Example');


>> HM_boxon = uimenu(Hm_top,...
```

```
                    'Label','Set' Box on',...
                    'CallBack',[...
                    'set(gca,''Box'',''on''), ' ,...
                    set(Hm_boxon,''Enable'',''off''), ' ,...
                    set(Hm_boxoff,''Enable'',''on'') ' ];

>> Hm_boxoff = uimenu(Hm_top,...
                    ' Label ' , ' Set Box off ' ,...
                    ' Enable ' , ' off ' ' , ' ,...
                    ' CallBack ' ,[...
                    ' set(gca,''Box'',''off'', ' ,...
                    ' set(Hm_boxon,''Enable'',''on''), ' ,...
                    ' set(Hm_boxoff''Enable'',''off'') ' ];
```

上例中引出了关于回调函数的另一个重点，即在变量 Hm_boxoff 定义之前，在回调字符串中用 Hm_boxoff 替代 Hm_boxon。因为回调字符串只是一个字符串，Matlab 不会给出警告，而且仅在 uimenu 被激活并将字符串传给 eval 时才由 Matlab 执行。

10.2.3 uicontrol 对象

控件是用户和应用程序进行交互的主要渠道，在 Matlab 中 uicontrol 是所有 GUI 控件的父类，而 uicontrol 对象主要有以下派生对象。

(1) 命令按钮(Push Button)。该控件主要用于将系统的控制转向某一个程序，以执行某种功能，如"OK(确定)"、"Cancel(取消)"等。

(2) 单选按钮(Radio Button)。该控件允许用户从多种选择项中选择其中的某一项内容。单选按钮只能进行"单项选择"的操作。

(3) 检查框(Check Box)。该控件允许用户从多种选择中选择其中的某些内容。被选中的检查框在其方形外框中有一个对号("√")标记。这种按钮主要用于让用户选用或不选用某种设置内容。检查框可以进行"多项选择"的操作。

(4) 列表框(List Box)。该控件用于从以列表的形式给出的一些项目中选择出其中的某一项内容，选中的项目将会出现在列表框的最上一行。

(5) 下拉式菜单(PoPupMenu)。该控件用于从系统所弹出的由一些命令按钮所组成的菜单中选取某一菜单项并执行相应的功能。

(6) 滑动条(Slider)。该控件可以以图示的方式使用户从某一范围的数值中选取某一个数值，该数值的大小通过滑动杆中滑块的位置来近似显示。

用户可以简单地使用 Matlab 提供的图形用户界面开发工具 GUIDE 来向一个窗体中添加这些控件，这个过程与目前大多数的 RAD 工具中的方法都是相似

的，用户可以根据自己的需要使用拖放的方式将工具栏中的控件添加到程序中，并可以很方便地设置这些控件的属性，同时可以编写控件事件响应函数。GUIDE的用法将在本章后面详细介绍。

图10.10所示为一个简单的例子，其中包含了Matlab提供的所有基本控件。

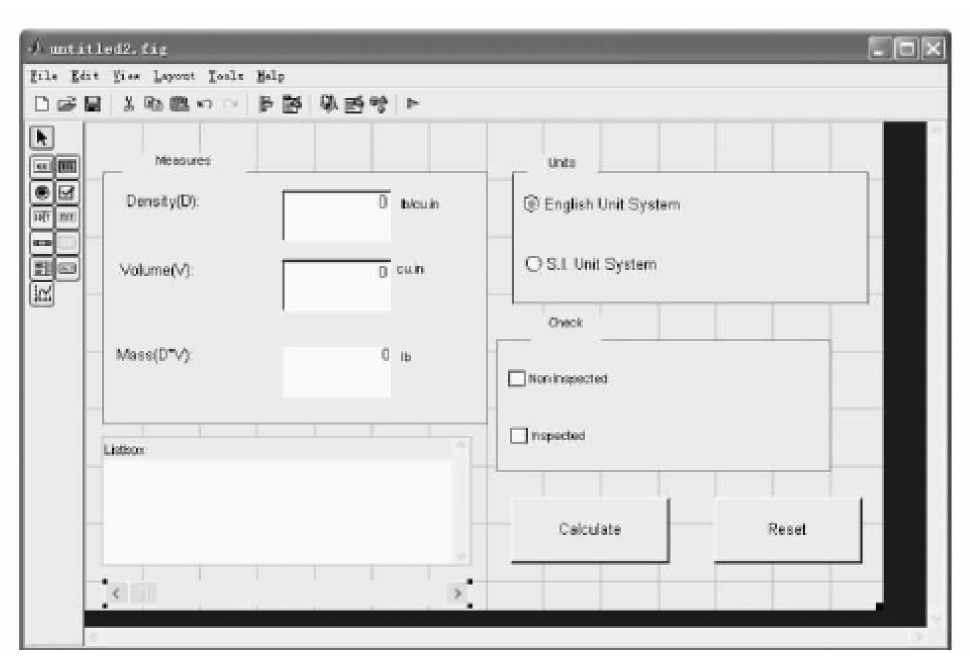


图10.10 Matlab GUI控件简单示例

由于Matlab GUI应用程序中使用的所有控件都是从uicontrol对象派生出来的，因此有必要了解这些控件的共同特性，表10.3列出了Uicontrol对象的属性及其含义。

其中带有*的属性为非文件式的，由{}括起来的属性值是默认值。

表10.3 Uicontrol对象属性列表

属 性	说 明
BackgroundColor	uicontrol背景色。3元素的RGB向量或Matlab一个预先定义的颜色名称。默认的背景色是浅灰色
Callback	Matlab回调串，当uicontrol激活时，回调串传给函数eval；初始值为空矩阵
ForegroundColor	uicontrol前景(文本)色。3元素的RGB向量或Matlab一个预先定义的颜色名称。默认的文本色是黑色
HorizontalAlignment	标志串的水平排列
left：	相对于uicontrol文本左对齐
{center}：	相对于uicontrol文本居中
right：	相对于uicontrol文本右对齐
Max	属性Value的最大许可值，最大值取决于uicontrol的Type当uicontrol处于on状态时，无线按钮及检查框将Value设定为Max；该值定义了弹出式菜单最小下标值或滑标的最大值。当Max－Min>1时，可编辑文本框是多行文本。默认值为1

Min	属性 Value 的最小许可值，最小值取决于 uicontrol 的 Type。uicontrol 处于 off 状态时，无线按钮及检查框将 Value 设定为 Min；该值定义了弹出式菜单最小下标值或滑标的最小值。当 Max－Min>1 时，可编辑文本框是多行文本。默认值为 0

（续）

属 性	说 明
Position	位置向量[left bottom width height]。其中，[left height]表示相对于图形对象左下角的 uicontrol 的左下角位置。[widht height]表示 uicontrol 的尺寸大小，其单位由属性 Units 确定
Enable *	控制框使能状态
〈on〉：	uicontrol 使能。激活 uicontrol，将 Callback 字符串传给 eval
off：	unicontrol 不使能，标志串模糊不清。激活 unicontrol 不起作用
String	文本字符串，在按钮键、无线按钮、检查框和弹出式菜单上指定 uicontrol 的标志。对于可编辑文本框，该属性设置成由用户输入的字符串。对弹出式菜单或可编辑文本框中多个选项，每一项用垂直条(|)分隔，整个字符串用引号括起来。框架和滑标不用引号
Style	定义 uicontrol 对象的类型
〈pushbutton〉：	按钮键：选择时执行一个动作
radiobutton：	无线按钮键：单独使用时，在两个状态之间切换；成组使用时，让用户选择一个选项
checkbox：	检查框：单独使用时，在两个状态之间切换；成组使用时，让用户选择一个选项
edit：	可编辑框：显示一个字符串并可让用户改变
text：	静态文本框：显示一个字符串
slider：	滑标：让用户在值域范围内可选择一个值
frame：	框架：显示包围一个或几个 uicontrol 的框架，使其形成一个逻辑群
popumenu：	弹出式菜单 ：含有许多互斥的可选择的弹出式菜单
Units	位置属性值的单位
inches：	英寸
centimeters：	厘米
normalized：	归一化的坐标值，图形的左下角映射为[0 0]而右上角的映射为[1 1]
points：	打印设置点，等于 1/72 英寸
〈pixels〉：	屏幕的像素(计算机屏幕分辨率的最小单位)
Value	uicontrol 的当前值。无线按钮和检查框在“on”状态时，value 设为 Max，当是“off”状态时，value 设为 Min。由滑标将滑标的 value 设置为数值(Min≤Value≤Max)，弹出式菜单把 value 值设置为所选择选项的下标(1≤Value≤Max)。文本对象和按钮不设置该属性
ButtonDownFcn	当 uicontrol 被选择时，Matlab 回调串传给函数 eval。初始值为空矩阵
Children	Uicontrol 对象一般无子对象，通常返回空矩阵
Clipping	限幅模式

〈on〉：	对 uicontrol 对象有作用效果
off：	对 uicontrol 对象无作用效果
DestroyFcn	没有文件说明（只对 Macintosh 4.2 版本）

（续）

属 性	说 明
Interrruptible	指定 ButtonDownFcn 和 CallBack 串是否可中断
〈on〉：	回调不能由其他回调中断
off：	回调可被中断
Parent	包含 uicontrol 对象的图形句柄
* Select	值为[on|off]
* Tag	文本串
Type	只读对象辨识串，通常为 uicontrol
UserData	用户指定的数据。可以是矩阵、字符串等
Visible	uicontrol 对象的可视性
〈on〉：	uicontrol 对象在屏幕上可见
off：	uicontrol 对象不可见，但仍然存在

下面是一个复杂一些的例子，借以说明 Uicontrol 各种控件的用法，这段代码制作一个能绘制任意图形的交互界面，它包括：可编辑文本框、弹出框、列表框。程序还将说明如何使编辑框允许输入多行指令。

```
% main.m
clf reset
set(gcf,'unit','normalized','position',[0.1,0.4,0.85,0.35]);    %设置图形窗大小
set(gcf,'defaultuicontrolunits','normalized');
set(gcf,'defaultuicontrolfontsize',11);
set(gcf,'defaultuicontrolfontname','隶书');
set(gcf,'defaultuicontrolhorizontal','left');
set(gcf,'menubar','none');                                       %删除图形窗工
具条
str='通过多行指令绘图的交互界面';
set(gcf,'name',str,'numbertitle','off');                         %书写图形窗名
h_axes=axes('position',[0.05,0.15,0.45,0.70],'visible','off');%定义轴位框位置
uicontrol(gcf,'Style','text',...                                 %制作静态文本框
  'position',[0.52,0.87,0.26,0.1],...
  'String','绘图指令输入框');
hedit= uicontrol(gcf,'Style','edit',...                          %制作可编辑文
本框
  'position',[0.52,0.05,0.26,0.8],...
```

```
   'Max',2);                    %取 2,使 Max-Min>1,从而允许多行输入

hpop=uicontrol(gcf,'style','popup',...                    %制作弹出菜单
   'position',[0.8,0.73,0.18,0.12],...
   'string','spring|summer|autumn|winter');              %设置弹出框中选
项名
hlist=uicontrol(gcf,'Style','list',...              %制作列表框
   'position',[0.8,0.23,0.18,0.37],...
   'string','Grid on|Box on|Hidden off|Axis off',...  %设置列表框中选项名
   'Max',2);                         %取 2,使 Max-Min>1,从而允许多项选择
   <23>
hpush=uicontrol(gcf,'Style','push',...             %制作与列表框配用的按键
   'position',[0.8,0.05,0.18,0.15],'string','Apply');
set(hedit,'callback','calledit(hedit,hpop,hlist)');  %编辑框输入引起回调
set(hpop,'callback','calledit(hedit,hpop,hlist)');  %弹出框选择引起回调
set(hpush,'callback','calledit(hedit,hpop,hlist)'); %按键引起的回调

%calledit.m
function calledit(hedit,hpop,hlist)
ct=get(hedit,'string');                          %获得输入的字符串函数
vpop=get(hpop,'value');                          %获得选项的位置标识
vlist=get(hlist,'value');                        %获得选项位置向量
if~isempty(ct)                     %可编辑框输入非空时
   eval(ct')                       %运行从编辑文本框送入的指令
   <6>
   popstr={'spring','summer','autumn','winter'};   %弹出框色图矩阵
   liststr={'grid on','box on','hidden off','axis off'};%列表框选项内容
   invstr={'grid off','box off','hidden on','axis on'};%列表框的逆指令
   colormap(eval(popstr{vpop}))                   %采用弹出框所选色图
   vv=zeros(1,4);vv(vlist)=1;
   for k=1:4
   if vv(k);eval(liststr{k});else eval(invstr{k});end %按列表选项影响图形
   end
end
```

在 Matlab 命令行运行这个程序,然后在提示的命令输入框中输入需要执行的图形显示代码,就可以得到图 10.11 所示的结果。

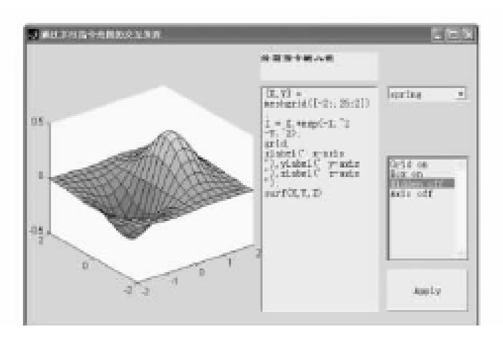


图 10.11　Matlab GUI 控件的另一个示例

10.3　脚本和回调函数

10.2 节介绍了 Matlab 中提供的各种 GUI 元素，本节将讲述如何对这些元素进行编程。GUI-M 文件就是用于控制 GUI 元素的代码，它决定了控件如何对用户事件进行响应，这种文件包含了所有运行 GUI 所需的代码，包括回调函数。GUIDE 开发环境为用户提供了开发 GUIM 文件的向导，用户只需将注意力集中在回调函数的编写上。

建立 GUI 函数的一个有效方法是编写独立的回调函数，专门执行一个或多个回调。函数使用的对象句柄和其他变量可以作为参量传递，必要时回调函数可返回值。

下例是以脚本文件建立一个方位角的滑标。

```
%setview.m script file

vw=get(gca,'View');

Hc_az=uicontrol(gcf,'Style','slider',...
            'Position',[10 5 140 20],...
            'Min',-90,'Max',90,'Value',vw(1),...
            'Callback',[...
                'set(Hc_cur,'String',num2str(get(Hc_az,'Value'))),' ...
                'set(gca,'View',[get(Hc_az,'Value') vw(2)])']);
Hc_min=uicontrol(gcf,'style','text',...
            'Position',[10 25 40 20],...
            'String',num2str(get(Hc_az,'Min')));

Hc_max=uicontrol(gcf,'Style','text',...
            'Position',[110 25 40 20],...
            'String',num2str(get(Hc_az,'Max')));
```

```
Hc_cur=uicontrol(gcf,'Style','text',...
                'Position',[60 25 40 20],...
                'String',num2str(get(Hc_az,'Value')));
```

下面是同样的例子。作为一个函数,采用 Tag 属性来辨别控制框,并使用独立的 M 文件来执行回调。

```
funtion setview( )
vw=get(gca,'View');
Hc_az=uicontrol(gcf,'Style','Slider',...
                'Position',[10 5 140 20],...
                'Min',-90,'Max',90,'Value',vw(1),...
                'Tag','Azslider',...
                'Callback','svcback');

Hc_min=uicontrol(gcf,'style','text,...
                'Position',[10 25 40 20],...
                'String',num2str(get(Hc_az,'Min')));

Hc_max=uicontrol(gcf,'Style','text',...
                'Position',[110 25 40 20],...
                'String',num2str(get(Hc_az,'Max')));

Hc_cur=uicontrol(gcf,'Style','text',...
                'Position',[60 25 40 20],...
                'Tag','Azcur',...
                'String',num2str(get(Hc_az,'Value')));
```

回调函数本身如下:

```
function svcback( )
vw = get(gca,'View');

Hc_az = findobj(gcf,'Tag','AZslider');
Hc_cur = findobj(gcf,'Tag' ,'AZcur');

str = num2str(get(Hc_az,'Value'));
newview =[get(Hc_az,'Value') vw(2)];
set(Hc_cur,'String',str)
set(gca,'View',newview)
```

上面的例子并不节省很多代码,但却得到了用函数而不用脚本文件的优点:回调函数可以利用临时变量,而不致使命令窗口工作空间拥挤;不需要 eval 所需的

引号和字符串;在回调函数中命令的句法变得十分简单。使用独立回调函数技术,越复杂的回调(函数)越简单。

独立回调函数的缺点是:需要很大数目的 M 文件以实现一个含有若干控制框和菜单项的 GUI 函数,所有这些 M 文件必须在 Matlab 路径中可得,且每一个文件又必须有一个不同的文件名。在对文件名大小有限制且对大小写不敏感的平台上,如 MS-Windows,文件冲突的机会就增加了。而且回调函数只能被 GUI 函数调用而不能被用户调用。

10.3.1 全局变量

全局变量可用在函数中,使某些变量对 GUI 函数的所有部分都可用。全局变量在函数的公共区说明,因此整个函数以及所有对函数的递归调用都可以利用全局变量。下例说明如何利用全局变量传递控件的图柄,从而保证回调动作正确执行。

(1) 主函数的代码:

```
% main.m
global hedit hpo phlist


clf reset,shg
set(gcf,'unit','normalized','position',[0.1,0.4,0.85,0.35]);    %设置图形窗大小
set(gcf,'defaultuicontrolunits','normalized');
set(gcf,'defaultuicontrolfontsize',11);
set(gcf,'defaultuicontrolfontname','隶书');
set(gcf,'defaultuicontrolhorizontal','left');
set(gcf,'menubar','none');                                       %删除图形窗工
具条
str='通过多行指令绘图的交互界面';
set(gcf,'name',str,'numbertitle','off');                         %书写图形窗名
h_axes=axes('position',[0.05,0.15,0.45,0.70],'visible','off');%定义轴位框位置
uicontrol(gcf,'Style','text',...                                 %制作静态文本框
  'position',[0.52,0.87,0.26,0.1],...
  'String','绘图指令输入框');
hedit= uicontrol(gcf,'Style','edit',...                          %制作可编辑文
本框
  'position',[0.52,0.05,0.26,0.8],...
  'Max',2);                      %取 2,使 Max-Min>1,从而允许多行输入
hpop=uicontrol(gcf,'style','popup',...                           %制作弹出菜单
  'position',[0.8,0.73,0.18,0.12],...
```

```
'string','spring|summer|autumn|winter');                  %设置弹出框中选项名
hlist=uicontrol(gcf,'Style','list',...                    %制作列表框
  'position',[0.8,0.23,0.18,0.37],...
  'string','Grid on|Box on|Hidden off|Axis off',...       %设置列表框中选项名
  'Max',2);                     %取 2,使 Max-Min>1,从而允许多项选择
  <23>
hpush=uicontrol(gcf,'Style','push',...                    %制作与列表框配用的按键
  'position',[0.8,0.05,0.18,0.15],'string','Apply');

set(hedit,'callback','calledit');                         %编辑框输入引起回调
set(hpop,'callback','calledit');               %弹出框选择引起回调
set(hpush,'callback','calledit');              %按键引起的回调
```

注意，在这段代码的最后三行，主函数向回调函数 calledit 传递了三个控件句柄，使得子函数可以使用这些全局变量。

（2）calledit 的代码：

```
% calledit.m
function calledit( )
global hedit hpophlist

ct=get(hedit,'string');                        %获得输入的字符串函数
vpop=get(hpop,'value');                        %获得选项的位置标识
vlist=get(hlist,'value');                      %获得选项位置向量
if ~isempty(ct)                                %可编辑框输入非空时
  eval(ct')                                    %运行从编辑文本框送入的指令
  <6>
  popstr={'spring','summer','autumn','winter'};      %弹出框色图矩阵
  liststr={'grid on','box on','hidden off','axis off'};%列表框选项内容
  invstr={'grid off','box off','hidden on','axis on'};%列表框的逆指令
  colormap(eval(popstr{vpop}))                           %采用弹出框所选色图
  vv=zeros(1,4);vv(vlist)=1;
  for k=1:4
    if vv(k);eval(liststr{k});else eval(invstr{k});end  %按列表选项影响图形
  end
end
```

这个函数通过全局的 GUI 句柄获得输入信息，并针对这些信息进行处理，然后将结果返回给所在的窗体。这段程序的结果与本章前面介绍的绘图命令的交互界面的结果即图 10.11 是一致的，在此就不再列出了。

10.3.2 递归函数调用

利用单独的M文件并递归地调用该文件，既可以避免多个M文件的复杂性，又可以利用函数的优点。使用开关switches或if elseif语句，可将回调函数装入调用函数内。通常这样一种函数调用的结构为：

```
function guifunc(switch)
```

其中switch确定执行哪一个函数开关的参量，它可以是字符串“startup”、“close”、“sectolor”等，也可以是代码或数字。如果switch是字符串，则可如下面所示的M文件片段那样将开关编程：

```
if nargin < 1, switch = 'startup'; end;
if ~isstr(switch), error('Invalid argument'), end;
if strcmp(switch,'startup'),
   <statement to create controls or menus>
   <statements to implement the GUI function>
elseif strcmp(switch,'setcolor'),
   <statements to perform the Callback associated with setcolor>
elseif strcmp(switch,'close'),
   <statements to perform the Callback associated with close>
end
```

如果是代码或字符串，开关也可以相同方式编程：

```
if nargin < 1,switch = 0;end;
if isstr(switch),error('Invalid argument'),end;
if switch = = 0,
   <statements to create controls or menus>
   <statements to implement the GUI function>
elseif switch = =1,
   <statements to perform the Callback associated with setcolor>
elseif switch ==2,
   <statements to perform the Callback associated with close>
end
```

下例说明了方位角滑标如何可作为单独的函数M文件来实现：

```
function setview(switch)
if nargin < 1,switch = 'startup';end;
if ~isstr(switch),error('Invalid argument.');end;

vw = get(gca,'view'); % This information is needed in both sections
```

```
if strcmp(switch,'startup') % Define the controls and tag them

   Hc_az = uicontrol(gcf,'Style','slider',...
                'Position',[10 5 140 20],...
                'Min',-90,'Max',90,'Value' vw(1),...
                'Tag','AZslider',...
                'Callback','setview('set')');

Hc_min=uicontrol(gcf,'Style','text',...
                'Position',[10 25 40 20],...
                'String',num2str(get(Hc_az,'Min')));
Hc_max = uicontrol(gcf,'Style','text',...
                'String',num2str(get(Hc_az,'Max')));

Hc_cur =uicontrol(gcf,'Style','text',...
                'Position',[60 25 40 20],...
                'Tag ','AZcur',...
                'string',num2str(get(Hc_az,'Value')));

elseif strcmp(switch,'set') % Execute the Callback

    Hc_az=findobj(gcf,'Tag','AZslider');
    Hc_cur-findobj(gcf,'Tag','AZcur');

    str = num2str(get(Hc_az,'Value'));
    newview - [get(Hc_az,'Value') vw(2)];

    set(Hc_cur,'String',str)
    set(gca,'View',newview)
end
```

上述的两个例子均设置了 tag 属性，利用该属性和函数 findobj 寻找回调函数所需对象的句柄。

10.4 GUIDE 的使用

GUIDE 是 Matlab 自带的图形界面辅助开发工具，它提供的模板和向导为用户开发 GUI 程序带来了很大的便利，本章将介绍这个工具的详细用法。

10.4.1 布局编辑器

GUIDE 提供的布局编辑器(Layout Editor)主要是针对 uicontrol 对象的排列和规划,它包括组件面板和窗格编辑器两部分,前面的章节中也有介绍,布局编辑器实现的功能与某些高级语言的 RAD 工具的界面规划部分相似。

启动布局编辑器需要在 Matlab 主命令行窗口中使用 guide 指令调出 GUI 环境,GUI 环境提供四种模板以供选择,分别是空模板、带 uicontrol 的模板、带坐标轴和菜单的模板以及普通的模式对话框。双击"GUI with uicontrol"后,就可以打开一个如图 10.12 所示的设计窗口。

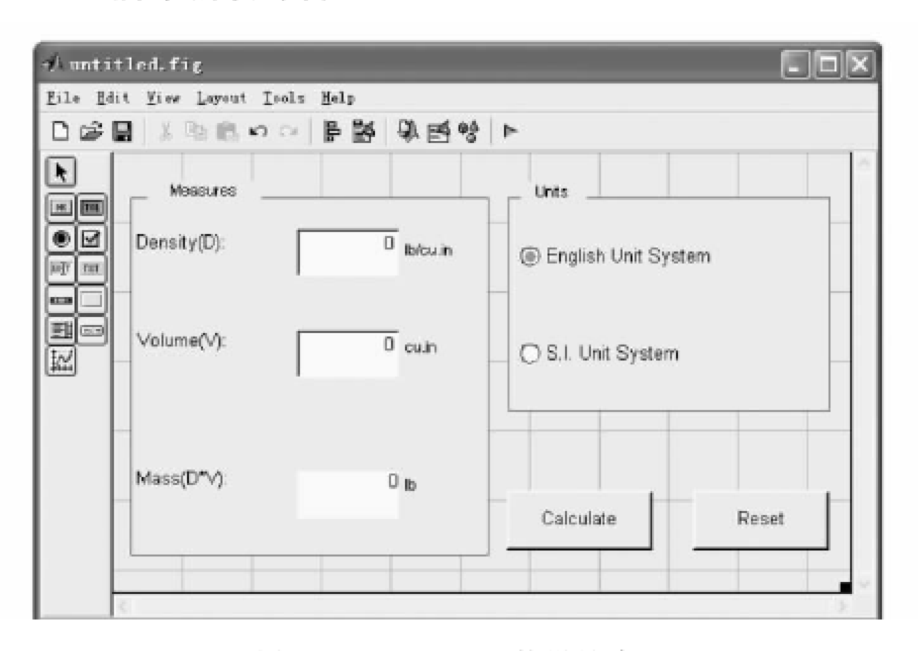


图 10.12 GUIDE 的设计窗口

打开布局编辑器的同时也创建了一个 GUI 文件,其后缀名为 fig。如果需要对一个创建好的文件进行编辑,可以直接在 Matlab 主命令窗口中输入下面的指令:

```
guide filename.fig
```

或者在编辑窗口中使用 open 菜单打开 fig 文件。除此之外还可以使用 openfig、open、hgload 指令来打开 fig 文件,例如,下面这些语句的作用是相同的:

```
% usage of openfig
openfig('filename.fig','new')
openfig('filename.fig','reuse')
openfig('filename.fig')
openfig('filename.fig','new','invisible')
openfig('filename.fig','new','visible')
figure_handle = openfig(...)
```

```
% usage of open
open('filename')

% usage of hgload
h = hgload('filename')
[h,old_props] = hgload(...,property_structure)
h = hgload(...,'all')
```

布局编辑器提供了测试环境，在工具栏点击“Run”按钮，编辑器会提示用户保存当前的工程，包括 fig 文件和 M 文件，同时设置好相应的运行目录，这样就可以测试运行 GUI 程序了。

布局编辑器除了包括编辑功能外，还包含了 GUIM 函数的编写向导，上下文菜单为每一个控件提供了 Callback 函数的入口。如图 10.13 中，选中按钮然后点击右键调出上下文菜单，可以看到“View Callbacks”选项，这个选项中包含了该 GUI 元素所能响应的所有事件，如选择 ButtondownFcn，就可以将处理代码加入到 GUIM 函数中。

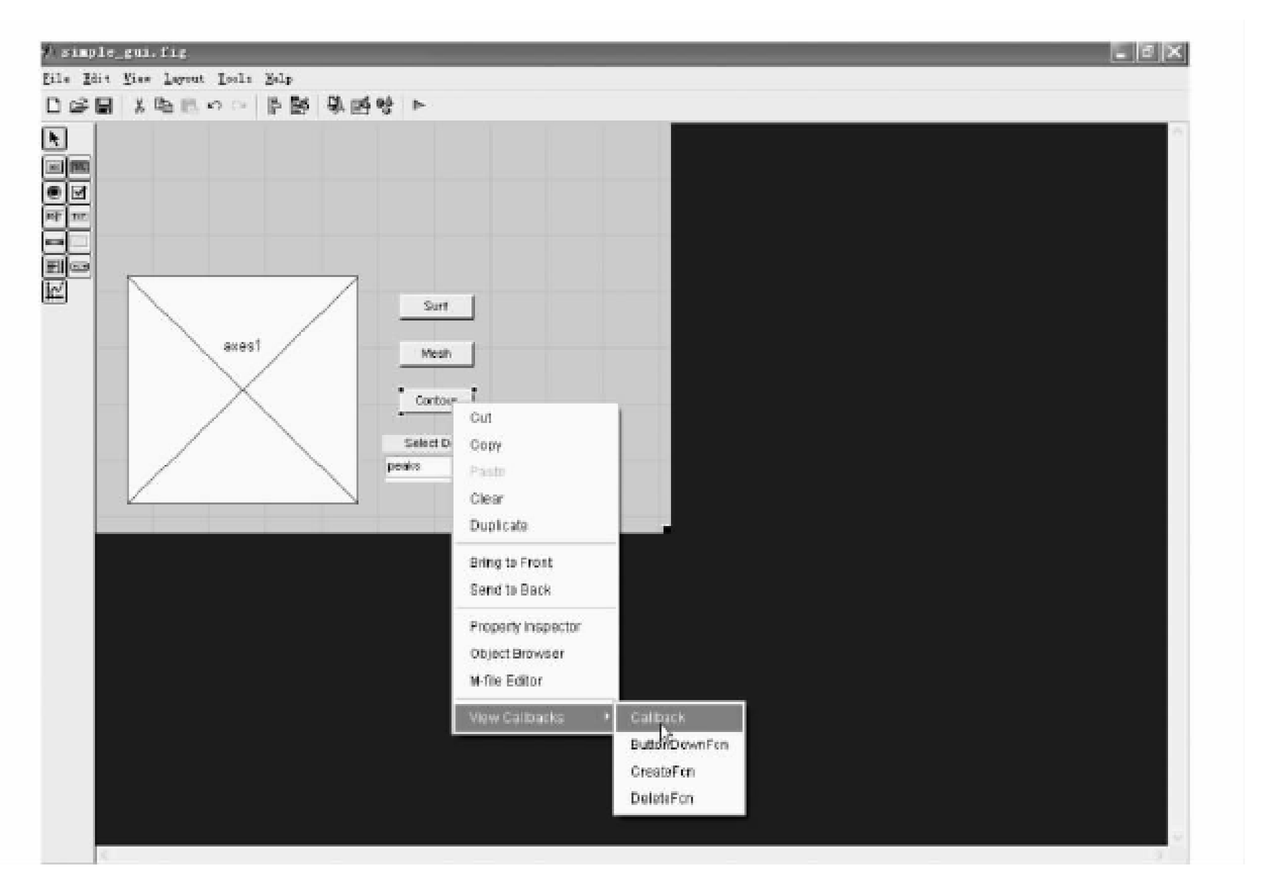


图 10.13　调用 GUIM 函数的编写向导

在对控件进行布局时，GUIDE 还提供了水平方向和竖直方向的对齐功能，这些可以借助标尺来完成，也可以由 GUIDE 自动完成，对齐的最小单位在两个方向上均是 9 个像素。

10.4.2 查看信息对象和编辑菜单

和其他高级语言的集成开发环境一样,GUIDE也提供了单个对象信息的查看,这里称为Property Inspector,在GUIDE中双击任意一个界面元素都可以调出如图10.14所示的这个窗口。

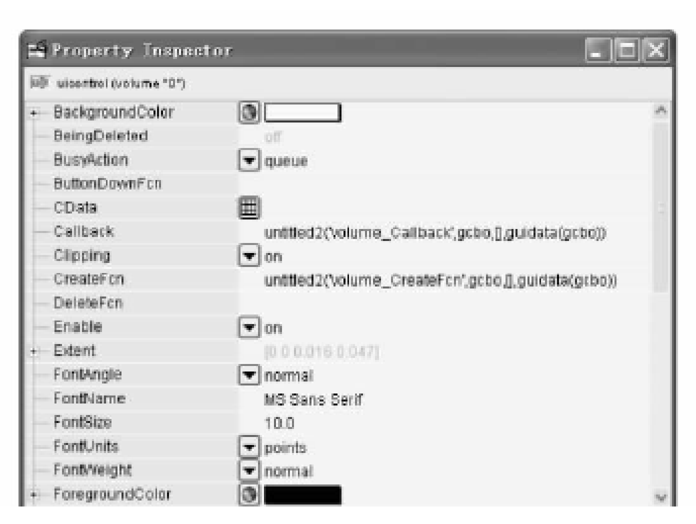


图 10.14 Property Inspector 窗口

Property Inspector 可以和对象浏览器以及 M 文件编辑器结合起来使用(见图10.15),选择相应的对象,就可以方便地在Property Inspector修改其属性,指定对象同时就可以在M文件编辑器中编写对应的处理例程。这几个工具都可以通过GUIDE主菜单的View菜单调出,但平时这几个菜单的属性是undockable的,因此都是以浮动窗口的形式出现。



图 10.15 对象浏览器和 M 文件编辑器结合使用

如果要处理菜单对象,则需要调出菜单编辑器。图10.16较为详细地说明了菜单编辑器的用法。

Matlab提供的菜单编辑器可以像Visual Basic和Delphi那样直观地创建出多级菜单,并且能够动态地对它们进行调整,还能对菜单的事件函数进行编辑。上

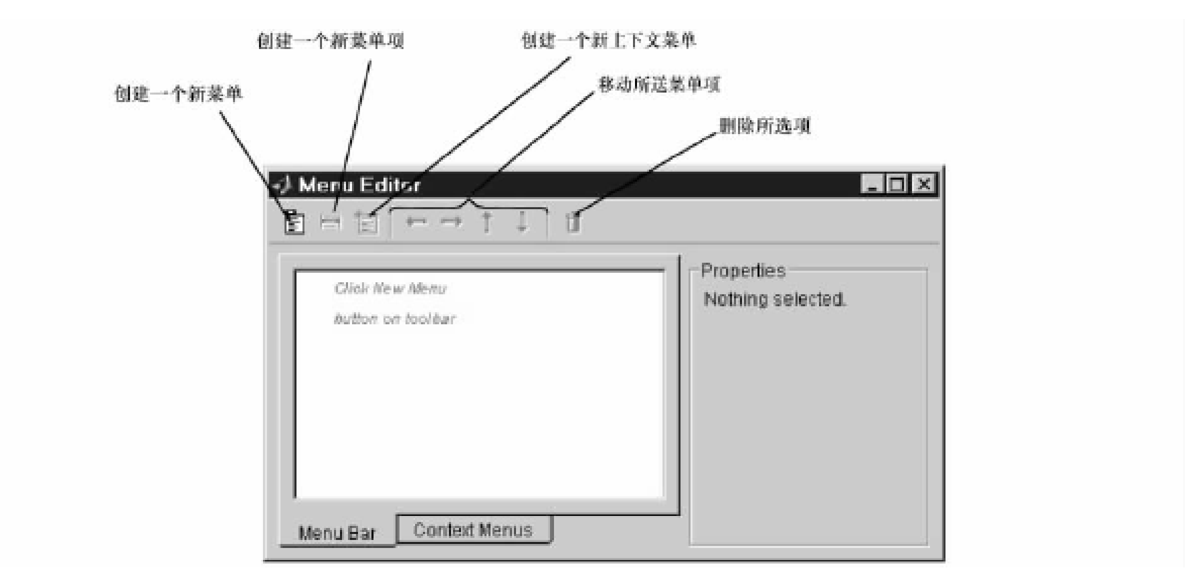


图 10.16　Matlab GUI 的菜单编辑器

下文菜单也可以在这个编辑器中实现。

10.5　Matlab GUI 综合实例

本节的综合示例是基于 Fourier-Mellin(傅里叶—梅林)变换算法的图像匹配程序。图像匹配是图像处理中的一个重要任务，它可以得到两幅图像中的重复部分。在这里利用了傅里叶变换的性质，这种算法中，对一个给定尺寸的图像，计算匹配参数的时间是固定的，这种算法的时间代价比较小。在匹配算法中要进行三次傅里叶变换和三次傅里叶反变换。

这个实例共包括以下主要文件：

(1) fm_gui.m，绘制图像界面的函数；

(2) fm_guifun.m，定义事件响应的函数和一些附加函数；

(3) hipass_filter.m，对图像进行滤波，去噪声；

(4) fourier_mellin.m，通过傅里叶变换进行图像匹配；

(5) fm_parse_inputs.m，从 data 数据对象中得到图像的参数。

这个实例的 GUI 界面如图 10.17 所示。

这个界面在文件 fm_gui.m 中定义，其中使用到三个坐标轴对象以及众多的下拉菜单和按钮对象。fm_gui.m 的主要工作就是对程序界面中用到的所有 GUI 对象进行静态描述和初始化工作，这个函数也是整个程序的入口，通常比较大的 GUI 程序的代码组织也是类似的。由于这个文件的代码比较简单，在这里就不列出来了。

函数 fm_parse_inputs.m 从 data 数据对象中得到图像的参数，这个函数有众多的输出参数，主要是两个输入的图形对象句柄和一些全局变量。这个函数还负责一些输入的错误检测，下面列出它的部分代码。

图 10.17　Fourier-Mellin 变换的 GUI 界面

```
function[input1,input2,ROT_METHOD,SCALE_METHOD,W
INDOW_TYPE,DISP_TEXT,PERF_LEVEL,WINDOW_SCALE]=fm_parse_inputs
(data)
% fm_parse_inputs(data)
%
input1=data.input1; % send in non zero-padded images means changing this
input2=data.input2;
%input1_windowed=data.input1_windowed;
%input2_windowed=data.input2_windowed;
ROT_METHOD=data.RotInterp;
SCALE_METHOD=data.SclInterp;
WINDOW_TYPE=data.windowType;
DISP_TEXT=data.dispText;
PERF_LEVEL=data.performanceLevel;
WINDOW_SCALE=data.windowScale;
```

函数 fm_guifun.m 是主要的 GUI 事件处理的代码：

```
function fm_guifun(action)
% GUI functions for Fourier-Mellin transform GUI
%
% Colorma p
m = gray(256);

switch(action)
```

```
  % Init %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case('create')
  data.pathname = ''; % path pointing to data

  % -------------------- window handle
storage---------------------
  data.hmain = get(findobj(gcbf,'Tag','Fig1'));
  % ---------- IMAGE 1 ---------------------
--------------------------------

  data.input1reference = [];
  data.input1 = []; % image data
  data.input1_windowed = [];
  data.input1_freq = [];
  data.input1_magSpec = [];
  data.input1_freq_lp= [];
  data.windowed_input1_freq_l P= [];
  data.logmagspec_lp_ms1 = [];
  data.filename1 = []; % filename corresponding to image 1

  % ---------- IMAGE 2 ---------------------
-----------------------------------

  data.input2reference = [];
  data.input2 = [];
  data.input2_windowed = [];
  data.input2_freq = [];
  data.input2_magSpec = [];
  data.input2_freq_lp= [];
  data.windowed_input2_freq_lp= [];
  data.logmagspec_lp_ms2 = [];
  data.filename2 = []; % filename corresponding to image 2

  % --------- SOME FOURIER-MELLIN PARAMETERS ------
------------------

  data.logpolarScaleRes = 256; % arguments for imlogpolar() function- they
control resolution of the log-polar plot
  data.logpolarAngleRes = 256;
```

```
  data.autocrop= 0; % automatically cro pinputs after resizing
  data.windowType = 'none'; % default window type
  data.RotInterp= 'nearest'; % the default interpolations to use
  data.SclInterp= 'nearest';
  data.LogInterp= 'nearest';
  data.dispText = 0;
  data.performanceLevel = 1;
  data.windowScale = 0;

  % --------- REGISTERED IMAGE ------------
-------------------------------

  data.registered = []; % registered image matrix
  data.input1registered = [];
  data.input2registered = [];
  data.pc_rs = []; % phase correlation for the log-polar form
  data.pc_trans = [];

  set(gcbf,'Userdata',data);
  set(findobj(gcbf,'Tag','CropInput2'),'String',[num2str(100) '%']);

  % Load image 1 %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

case('loadA')
  dispText('','b');
  data = get(gcbf,'Userdata');
  pathname = data.pathname;

  dispTag('Ref_im','r'); % this stuff isn't that apparent in the aplication??!!
  dispText('Loading image 1','b');
  [filename, pathname] = uigetfile([pathname '*.*'], 'Load image 1'); %
GUI file browser
  if filename~=0 % if we have a file
    if isempty(findstr(filename,'pgm')) % if not a PGM
      [M1,ma] = imread([pathname, filename]);
      if isind(M1) & ~isempty(ma)
        M1 = 256 * double(ind2gray(M1,ma));
      else
```

```
            if isgray(M1)
               M1 = double(M1);
            else
               M1 = double(rgb2gray(M1));
            end;
         end;
   else % if it is a PGM
         cesta=strrep([pathname, filename],'.pgm',''); % stri poff the .pgm
bit for some reason
M1=readpgm(cesta); % special pgm reader?!!
   end;

   data.input1reference = M1;
   data.input1 = M1;
   data.input1_windowed = window2d(size(M1,1),size(M1,2),
data.windowType).* M1;
   set(gcbf,'Userdata',data);
   updateImage(1,0); % update all the other plots...
   data = get(gcbf,'Userdata');

   imDims = size(M1); % dimensions

   set(findobj(gcbf,'Tag','Ref_im_c'),'String',[filename ', ' int2str(imDims(1))
' x ' int2str(imDims(2))],'ForegroundColor','k');

   data.pathname = pathname; % save pathname of this file

   data.filename1 = filename;
   set(gcbf,'Userdata',data);
   dispTag('Ref_im','k');
   dispText('','b');

  end;

% Load image 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
……
……
```

```
……
……

% * * * * * * * * * * * * * * * * * * * * * *
 * * * * * * * auxilliary functions

function dispText(txt,colr);

set(findobj(gcbf,'Tag','MessText'),'String',txt,'ForegroundColor',colr);



function vr=vrat(ktery);

idx = get(findobj(gcbf,'Tag',ktery),'value');
val = get(findobj(gcbf,'Tag',ktery),'String');
vr=str2num(val(idx));

function ramek(kde,barva);

set(findobj(gcbf,'Tag',kde),'XColor',barva);
set(findobj(gcbf,'Tag',kde),'YColor',barva);
set(findobj(gcbf,'Tag',kde),'ZColor',barva);

function dispTag(txt,colr);

set(findobj(gcbf,'Tag',txt),'ForegroundColor',colr);

function updateImage(im,LP_ONLY)
% USAGE: updateImage(im,LP_ONLY) A. I. Wilmer, 2002
%
% function to update magnitude spectra, log-polar plots etc of image 'im'
% LP_ONLY : only update the log-polar plot and stuff dependent on it

data = get(gcbf,'Userdata');

if (im==1) % then update image 1 information
   if (~LP_ONLY) % if log-polar setting is changed then don't need
to do the next couple of lines
```

```
        data.input1_freq = fftshift(fft2(data.input1_windowed));
        data.input1_magSpec = hipass_filter(size(data.input1_freq,1),
    size(data.input1_freq,2)).* abs(data.input1_freq);
    % data.input1_magSpec = log10(abs(data.input1_freq));
        end
        data.input1_freq_lp= imlogpolar(data.input1_magSpec,data.logpolarScaleRes,
    data.logpolarAngleRes,data.LogInterp);
        data.windowed_input1_freq_lp= repmat(window1d(size(data.input1_freq_lp,1),
    data.windowType),1,size(data.input1_freq_lp,2)).* data.input1_freq_lp;
        data.logmagspec_lp_ms1 = hipass_filter(size(data.input1_freq_lp,1),
    size(data.input1_freq_lp,2)).* abs(fftshift(fft2(data.input1_freq_lp)));
    elseif (im==2) % update image 2 plots
        if (~LP_ONLY) % if log-polar setting is changed then don't need to do the next
    couple of lines
        data.input2_freq = fftshift(fft2(data.input2_windowed));
        data.input2_magSpec = hipass_filter(size(data.input2_freq,1),
    size(data.input2_freq,2)).* abs(data.input2_freq);
    % data.input2_magSpec = log10(abs(data.input2_freq));
      end
      data.input2_freq_lp= imlogpolar(data.input2_magSpec,data.logpolarScaleRes,
    data.logpolarAngleRes,data.LogInterp);
      data.windowed_input2_freq_lp= repmat(window1d(size(data.input2_freq_lp,1),
    data.windowType),1,size(data.input2_freq_lp,2)).* data.input2_freq_lp;
      data.logmagspec_lp_ms2 = hipass_filter(size(data.input2_freq_lp,1),
    size(data.input2_freq_lp,2)).* abs(fftshift(fft2(data.input2_freq_lp)));
    else
      disp('updateImage(): incorrect image number used.')
    end
    set(gcbf,'Userdata',data);
```

上面的代码中省略了 load image2 部分，这一部分内容和 load image1 的相似。

整个函数首先使用 gray(256)函数定义了颜色映射类型，接着进行了一些初始化的工作，包括对各个对象句柄的初始化，如定义了各个菜单的初始值等，其中还保存了主窗口对象的句柄，使用的是下面的语句：

```
    data.hmain = get(findobj(gcbf,'Tag','Fig1'));
```

然后是 Fourier-Mellin 变换的一些参数的定义，包括旋转、放缩和极坐标变换选择的插值方式。这些参数将在 fourier_mellin.m 函数中用。

接着是与输出图像坐标轴相关的 GUI 的定义和初始化，并使用下面的语句将变换函数的输出和窗口的数据输出关联起来：

```
set(gcbf,'Userdata',data);
set(findobj(gcbf,'Tag','CropInput2'),'String',[num2str(100) '%']);
```

在装载图像的时候根据上面定义的全局参数传递给 uigetfile 函数，这些参数包括文件名和路径名等，再调用这个函数，结束后根据返回的结果不同进行处理，如果成功则将图像显示在第一个坐标轴对象中，否则转到错误处理的代码。

第二张用于对比匹配的图像的装载处理和第一张情况类似，这里不再重复说明。

最后是一些辅助函数，包括图像说明文字的处理和一些数据的转换等。

下面是程序的主要算法实现函数 fourier_mellin.m 的代码：

```
function [combImage,registered1,registered2,reg_output,cps_rs,cps_trans] =
fourier_mellin(data)
% USAGE : [combImage,registered1,registered2,reg_output,cps_rs,cps_trans] =
fourier_mellin(data)

global rho

[input1, input2, ROT__METHOD, SCALE__METHOD, WINDOW__TYPE, DISP__
TEXT,
SORTLIST,WINDOW_SCALE] = fm_parse_inputs(data);

PHASECORR_WINDOW = 0; % don't apply a window prior to
calculation phase-correlation
SCALE_THRESHOLD = 6;

% --------------------------
------------------------
% RECOVER ROTATION AND SCALE
% 3) Apply phase-correlation and recover rotation and scale
if (DISP_TEXT) disp('--------'); end
if (WINDOW_SCALE)
  disp('need to confirm whether this is a beneficial thing to do...')
  cps_rs                                                    =
crosspowerspectrum(data.windowed_input1_freq_lp,data.windowed_input2_freq_lp);
else
  cps_rs = crosspowerspectrum(data.input1_freq_lp,data.input2_freq_lp);
end
degrees_per_pixel = 360/size(cps_rs,2);
```

```
% - SORTING THE PHASE CORRELATION PEAKS
TO IMPLEMENT THE BODGE SLIDER ----------
sorted_cps_rs = sort(cps_rs(:)); % sort the phase correlation output so we
can list the most likely rotations/scale combinations (fast)

reg_output.trans_peak = -Inf;
FINEFLAG = 0;

for sorted_index = length(sorted_cps_rs):-1:(length(sorted_cps_rs)+
1-SORTLIST) % iterate through the highest peaks
  [irx,jrx] = find(cps_rs==sorted_cps_rs(sorted_index));              % find
{scale,rotation} corresponding to maximum point
sorted_cps_rs_peak = cps_rs(irx,jrx);
sorted_rotation = degrees_per_pixel * (jrx-1);
% decode the scale
if (irx > size(cps_rs,1)/2) % then input2 has been scaled DOWN wrt input1
  dsi = size(cps_rs,1)-irx+2; % not sure why the 2 but reference
images show it is necessary
else % input2 has been scaled UP wrt input1
  dsi = irx;
end

if(dsi<=1)
  scale_neighbourhood = [NaN; rho(dsi); rho(dsi+1)];
elseif (dsi>=length(rho))
  scale_neighbourhood = [rho(dsi-1); rho(dsi); NaN];
else
  scale_neighbourhood = [rho(dsi-1);rho(dsi); rho(dsi+1)];
end
if (irx > size(cps_rs,1)/2) % then input2 has been scaled DOWN wrt input1
  scale_neighbourhood = 1./scale_neighbourhood;
end
sorted_scale = scale_neighbourhood(2);

if (sorted_scale>(1/SCALE_THRESHOLD))&(sorted_scale<SCALE_THRESH-
OLD)
% a lot of scales are stupidly large so threshold them out
  sorted_rotRect1 = imrotate(input2,-sorted_rotation,ROT_METHOD,'crop');
```

```
sorted_rotRect2 = imrotate(input2, -(sorted_rotation + 180), ROT_METHOD,'
crop');
% rectify for rotation and 180degs plus rotation
    sorted_rsRect1 = imresize(sorted_rotRect1,1.0/sorted_scale,SCALE_METHOD);
sorted_rsRect2 = imresize(sorted_rotRect2,1.0/sorted_scale,SCALE_METHOD);
% rectify sorted image prior to translation registration

    [input1,sorted_rsRect1] = zeropad(input1,sorted_rsRect1,0);
% zero-pad the images to be the same size
    [input1,sorted_rsRect2] = zeropad(input1,sorted_rsRect2,0);
% zero-pad the images to be the same size
    sorted_cps_trans1 = crosspowerspectrum(input1,sorted_rsRect1);
sorted_cps_trans2 = crosspowerspectrum(input1, sorted_rsRect2);
% perform phase-correlation

    if(max(max(sorted_cps_trans1))>max(max(sorted_cps_trans2)));
% then don't add the 180
      [i1,j1]=find(sorted_cps_trans1==max(max(sorted_cps_trans1)));
      sorted_cps_trans_peak = sorted_cps_trans1(i1,j1);
      sorted_rsRect = sorted_rsRect1;
      cps_trans = sorted_cps_trans1;
    else% then add the 180
      [i1,j1]=find(sorted_cps_trans2==max(max(sorted_cps_trans2)));
      if (sorted_rotation<180)
        sorted_rotation = sorted_rotation+180;
      else
        sorted_rotation = sorted_rotation-180;
    end
      sorted_cps_trans_peak = sorted_cps_trans2(i1,j1);
      sorted_rsRect = sorted_rsRect2;
      cps_trans = sorted_cps_trans2;
    end

    %decode the translation
    sortedHtTrans = i1-1; sortedWdTrans = j1-1;
    if(i1>size(cps_trans,1)/2) % then need to read the height from the bottom
of the phase correlation plot
        sortedHtTrans = sortedHtTrans-size(cps_trans,1);
```

```
    end
    if (j1>size(cps_trans,2)/2) % then need to read the height from the right
of the phase correlation plot
        sortedWdTrans = sortedWdTrans-size(cps_trans,2);
    end

    if (sorted_cps_trans_peak>reg_output.trans_peak)
        FINEFLAG=1;
        reg_output.translation = [sortedHtTrans sortedWdTrans];
        reg_output.rotation = sorted_rotation;
        reg_output.scale = sorted_scale;
        reg_output.trans_peak = sorted_cps_trans_peak;
        reg_output.rs_peak = sorted_cps_rs_peak;
        the_one_to_use = sorted_rsRect;
        scales = scale_neighbourhood;
        if (DISP_TEXT) disp(['SAVING: Peak ',num2str(length(sorted_cps_rs)-
sorted_index+1),'at rotation=',num2str(sorted_rotation),', scale=',
num2str(sorted_scale),'(cps_rs peak ht=',num2str(sorted_cps_rs_peak),')
with MAX trans peak at (',num2str(sortedHtTrans),',',num2str(sortedWdTrans),')
with ht=',num2str(sorted_cps_trans_peak)]); end
          else
            if (DISP_TEXT) disp(['NOT SAVING: Peak ',num2str
(length(sorted_cps_rs)-sorted_index+1),'at rotation=',
num2str(sorted_rotation),', scale=',num2str(sorted_scale),'
(cps_rs peak ht=',num2str(sorted_cps_rs_peak),') with MAX trans peak at
(',num2str(sortedHtTrans),',',num2str(sortedWdTrans),')
with ht=',num2str(sorted_cps_trans_peak)]); end
          end
      end
end

if (~FINEFLAG)
      disp(['There is no solution as the recommended scale factor is ',
num2str(sorted_scale),' which is, quite frankly, ridiculous. Move the slider
to the
RIGHT and try again to look for more suitable solutions.'])
      reg_output.translation = [0 0];
      reg_output.rotation = 0;
```

```
        reg_output.scale = 1;
        reg_output.trans_peak = -Inf;
        reg_output.rs_peak = sorted_cps_rs_peak;
        the_one_to_use = input2;
end

% ------- DISPLAY SOME GENERAL INFO ABOUT THE
FOURIER-MELLIN
SOLUTION ------if (DISP_TEXT)
        disp(['Orientation neighbourhood is { ',num2str(reg_output.
rotation-degrees_per_pixel),', * ',num2str(reg_output.rotation),' * , ',
num2str(reg_output.rotation + degrees_per_pixel),' }, i.e., resolution of
',num2str
(degrees_per_pixel),' degrees']);
        disp(['Scale neighbourhood is {',num2str(scales(1)),', * ',
num2str(scales(2)),' * , ',num2str(scales(3)),'}'])
disp(['Translation is (',num2str(reg_output.translation(1)),',',
num2str(reg_output.translation(2)),')'])
end
% -------DO A RECONSTRUCTION FOR VISUAL PURPOSES
-----------------------


input2_rectified = the_one_to_use; move_ht = reg_output.translation(1);
move_wd = reg_output.translation(2);

total_height = max(size(input1,1),(abs(move_ht)+size(input2_rectified,1)));
total_width = max(size(input1,2),(abs(move_wd)+size(input2_rectified,2)));
combImage = zeros(total_height,total_width); registered1 =
zeros(total_height,total_width); registered2 = zeros(total_height,total_width);

% if move_ht and move_wd are both POSITIVE
if((move_ht>=0)&(move_wd>=0))
      registered1(1:size(input1,1),1:size(input1,2)) = input1;
registered2((1+move_ht):(move_ht+size(input2_rectified,1)),
(1+move_wd):(move_wd+size(input2_rectified,2))) = input2_rectified;
elseif ((move_ht<0)&(move_wd<0)) % if translations are both NEGATIVE
      registered2(1:size(input2_rectified,1),1:size(input2_rectified,2)) =
input2_rectified;
```

```
registered1((1+abs(move_ht)):(abs(move_ht)+size(input1,1)),
(1+abs(move_wd)):(abs(move_wd)+size(input1,2))) = input1;
elseif ((move_ht>=0)&(move_wd<0))

registered2((move_ht+1):(move_ht+size(input2_rectified,1)),
1:size(input2_rectified,2)) = input2_rectified;
    registered1(1:size(input1,1),(abs(move_wd)+1):(abs(move_wd)+
size(input1,2))) = input1;
elseif ((move_ht<0)&(move_wd>=0))
    registered1((abs(move_ht)+1):(abs(move_ht)+
size(input1,1)),1:size(input1,2)) = input1;
registered2(1:size(input2_rectified,1),(move_wd+
1):(move_wd+size(input2_rectified,2))) = input2_rectified;
end

if sum(sum(registered1==0)) > sum(sum(registered2==0)) % find the im-
age with the
greater number of zeros - we shall plant that one and then bleed in the
other for the combined image
    plant = registered1; bleed = registered2;
else
    plant = registered2; bleed = registered1;
end

combImage = plant;
for p=1:total_height
    for q=1:total_width
      if (combImage(p,q)==0)
        combImage(p,q) = bleed(p,q);
        end
      end
    end
```

整个算法在完成一次图像匹配的时候需要进行三次傅里叶变换和三次傅里叶反变换。设 input1 和 input2 是输入的进行匹配的图像，函数 fourier-mellin. m 对 input2 图像可以进行旋转和放缩，并匹配原始图像和旋转放缩后的图像。

运行整个程序的时候分别装入两张图像，并对 input2 旋转 4°。peak 选择 29，然后进行匹配远算，得到如图 10.18 所示的结果。

其中，Rotate 是旋转的角度，Scale 是放缩的尺寸，而 Log-po 是极坐标变换选

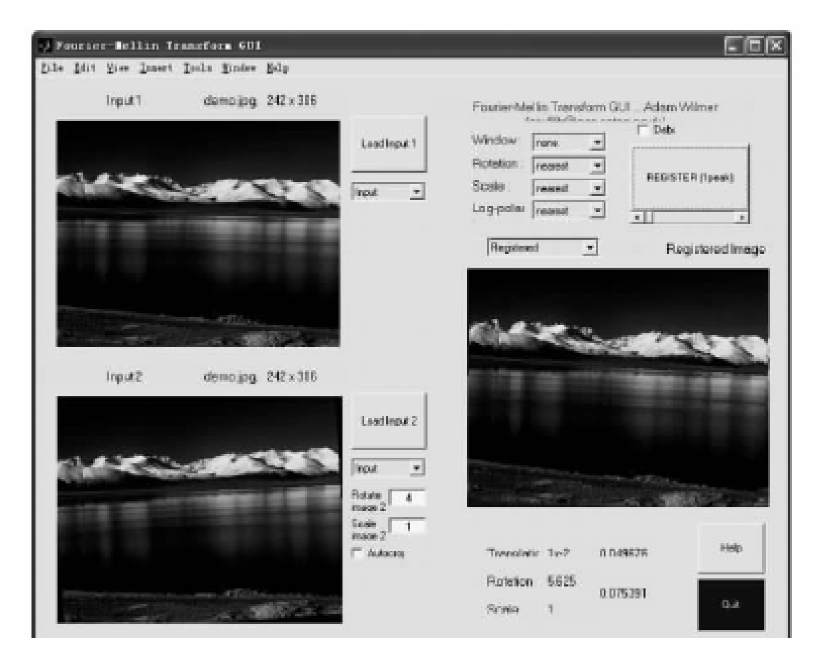


图 10.18　Fourier-Mellin 变换的演示结果

择的插值方式。Peak 值是在匹配中迭代的次数，选择越高，匹配越精确，但是计算的时间代价越大。右下角的 Rotation 等是对旋转、放缩大小的计算值。

关于这个算法的详细信息可以参看 http://www.ecs.soton.ac.uk/～aiw99r/fm_transform/中的相关内容，这里不再深入介绍。

附录　Matlab 图像处理工具箱函数

附表 1　图像显示

函 数	功　能	函 数	功　能
Colorbar	显示颜色条	Subimage	在一幅图中显示多幅图像
Getimage	从坐标轴获得图像数据	Truesize	调整图像显示尺寸
Imshow	显示图像	Warp	将图像显示到纹理映射表面
Montage	在矩形框中同时显示多幅图像	Zoom	缩放图像
Immovie	创建多帧索引图的电影动画		

附表 2　图像文件 I/O

函 数	功　能
Imfinfo	返回图形文件信息
Imread	从图形文件中读取图像
Imwrite	将图像写入到图形文件中

附表 3　几何操作

函 数	功　能
Imcorp	剪切图像
Imresize	改变图像大小
Imrotate	旋转图像

附表 4　像素和统计处理

函 数	功　能	函 数	功　能
Corr2	计算两个矩阵的二维相关系数	Improfile	计算图像中一条线段或多条线段强度（灰度）值，并绘制其图形
Imcontour	创建图像的轮廓图	Mean2	计算矩阵元素的平均值
Imfeature	计算图像区域的特征尺寸	Pixval	显示像素的数据值，并且能够显示两个像素间的欧几里得几何距离
Imhist	显示图像的直方图	Std2	计算矩阵元素的标准偏移
Impixel	显示像素的数据值		

附表 5　区域处理

函 数	功　能
Roicolor	选择感兴趣的颜色区
Roifill	在图像的任意区域中进行平滑插值
Roifilt2	过滤敏感区域
Roipoly	选择一个敏感的多边形区域

附表 6　图像分析

函 数	功　能
Edge	利用各种算子来作边缘检测
qtdecomp	实现指定图像的四叉树分解
Qtgetblk	得到四叉树分解后子块像素及其位置信息
Qtsetblk	设置四叉树分解中的块值

附表 7　图像增强

函 数	功　能
Histeq	实现对输入图像的直方图均衡化
imadjust	调整图像的灰度值或颜色映像表
imnoise	增强图像的渲染效果
Medfilt2	实现对指定图像的中值滤波
Ordfilt2	进行二维统计顺序滤波
Wiener2	进行二维适应性去噪过滤处理

附表 8　线性滤波

函 数	功　能
Conv2	进行二维卷积
Convmtx2	计算二维卷积矩阵
Convn	计算 n 维卷积
Filter2	进行二维线性过滤操作
Fspecial	创建一个指定的滤波器

附表 9　线性二维滤波设计

函 数	功　能	函 数	功　能
Freqspace	确定二维频率响应的频率空间	Ftrans2	通过频率转换设计二维 FRI 滤波器
Freqz2	计算二维频率响应	Fwind1	用一维窗口方法设计二维 FRI 滤波器
Fsamp2	用频率采样法设计二维 FRI 滤波器	Fwind2	用二维窗口方法设计二维 FRI 滤波器

附表 10　图像变换

函 数	功　能	函 数	功　能
Dct2	实现图像的二维离散余弦变换	Ifft2	计算二维傅里叶变换的反变换
dctmtx	计算二维 DCT 矩阵	ifftn	计算 n 维傅里叶变换的反变换
Fft2	计算二维快速傅里叶变换	Radon	计算图像 I 在指定角度的 Radon 变换
fftn	计算 n 维快速傅里叶变换	Iradon	计算 Radon 反变换
fftshift	将傅里叶变换后的图像频谱中心从矩阵的原点移动到矩阵的中心	Phantom	产生一个头部幻影图像
Idct2	实现图像的二维离散余弦反变换		

附表 11　边缘和块处理

函 数	功　能	函 数	功　能
bestblk	确定块操作的大小	Im2col	重新调整图像块为列
Blkproc	实现图像的块显示	Nlfilter	进行边沿操作
Col2im	将矩阵的列重新组织到块中		

附表 12　二进制对象操作

函 数	功　能	函 数	功　能
Makelut	创建查找表	Bwmorph	对图像作指定的形态运算
Applylut	在二进制图像中利用查找表进行边沿操作	Bwperim	对二值图像进行边界提取

（续）

函数	功能	函数	功能
Bwarea	计算二进制图像的面积	Bwselect	在二值图像中选择对象
bweuler	计算图像的欧拉数	Imdilate	对指定的图像作膨胀运算
imfill	对图像进行填充	Imerode	对指定的图像作腐蚀运算
Bwlabel	对二值图像中各个分离部分进行标记		

附表 13　颜色映像处理

函数	功能	函数	功能
Brighten	增加或降低颜色映像表的亮度	Imapprox	对索引图像进行近似处理
Cmpermute	调整颜色映像表中的颜色	Rgbplot	划分颜色映像表
Comunique	查找颜色映像表中特定的颜色和相应的图像		

附表 14　颜色空间转换

函数	功能	函数	功能
Hsv2rgb	将 HSV 值转换到 RGB 颜色空间	Rgb2ntsc	将 RGB 值转换到 NTSC 颜色空间
Ntsc2rgb	将 NTSC 值转换到 RGB 颜色空间	Rgb2ycbcr	将 RGB 值转换到 YcbCr 颜色空间
Rgb2hsv	将 RGB 值转换到 HSV 颜色空间	Ycbcr2rgb	将 YcbCr2 值转换到 RGB 颜色空间

附表 15　图像类型转换

函数	功能	函数	功能
Dither	通过抖动增加外观颜色分辨率	Int2rgb	将检索图像转换为 RGB 图像
Gray2ind	将灰度图像转换为索引图像	Isbw	判断是否为二进制图像
grayslice	从灰度图像创建索引图像	Isgray	判断是否为灰度图像
Im2bw	将图像转换为二进制图像	Isind	判断是否为索引图像
Im2double	将图像矩阵转换为双精度型	Isgrb	判断是否为 RGB 图像
Im2unit8	将图像矩阵转换为 8 位无符号整型	Mat2gray	将矩阵转换为灰度图像
Im2unit16	将图像矩阵转换为 16 位无符号整型	Rgb2gray	将 RGB 图像或颜色映像表转换为灰度图像
Ind2gray	将检索图像转换为灰度图像	Rgb2ind	将 RGB 图像转换为索引图像